Learning SHACL Shapes from Knowledge Graphs

Pouya Ghiasnezhad Omran [a,*], Kerry Taylor [a], Sergio Rodriguez Mendez [a], and Armin Haller [a]
[a] *School of Computing, The Australian National University , ACT, Australia*
E-mails: P.G.Omran@anu.edu.au, Kerry.Taylor@anu.edu.au, Sergio.RodriguezMendez@anu.edu.au, Armin.Haller@anu.edu.au

Abstract. Knowledge Graphs (KGs) have proliferated on the Web since the introduction of knowledge panels to Google search in 2012. KGs are large data-first graph databases with weak inference rules and weakly-constraining data schemes. SHACL, the Shapes Constraint Language, is a W3C recommendation for expressing constraints on graph data as shapes. SHACL shapes serve to validate a KG, to underpin manual KG editing tasks, and to offer insight into KG structure. Often in practice, large KGs have no available shape constraints and so cannot obtain these benefits for ongoing maintenance and extension.

We introduce Inverse Open Path (IOP) rules, a predicate logic formalism which presents specific shapes in the form of paths over connected entities. IOP rules express simple shape patterns that can be augmented with minimum cardinality constraints and also used as a building block for more complex shapes, such as trees and other rule patterns. We define formal quality measures for IOP rules and propose a novel method to learn high-quality rules from KGs. We show how to build high-quality tree shapes from the IOP rules. Our learning method, SHACLEARNER, is adapted from a state-of-the-art embedding-based open path rule learner (OPRL).

We evaluate SHACLEARNER on some real-world massive KGs, including YAGO2s (4M facts), DBpedia 3.8 (11M facts), and Wikidata (8M facts). The experiments show that SHACLEARNER can effectively learn informative and intuitive shapes from massive KGs. The shapes are diverse in structural features such as depth and width, and also in quality measures that indicate confidence and generality.

Keywords: SHACL Shape Learning, Shapes Constraint Language, Knowledge Graph, Inverse Open Path Rule

1. Introduction

While public knowledge graphs (KGs) became popular with the development of DBpedia [1] and Yago [2] more than a decade ago, interest in enterprise knowledge graphs [3] has taken off since the inclusion of knowledge panels on the Google Search engine in 2012, driven by its internal knowledge graph. Although these KGs are massive and diverse, they are typically incomplete. Regardless of the method that is used to build a KG (e.g., collaboratively vs individually, manually vs automatically), it will be incomplete because of the evolving nature of human knowledge, cultural bias [4] and resource constraints. Consider Wikidata [5], for example, where there is more complete information for some types of entities (e.g., pop stars), while less for others (e.g., opera singers). Even for the same type of entity, for example, computer scientists, there are different depths of detail depending on the country of residence of the scientist.

However, the power of KGs comes from their data-first approach, enabling contributors to extend a KG in a relatively arbitrary manner. By contrast, a relational database typically employs not-null and other schema-based constraints that require some attributes to be instantiated in a defined way at all times. Large KGs are typically populated by automatic and semi-automatic methods using non-structured sources such as Wikipedia that are prone to errors of omission (i.e., incompleteness) and commission (i.e., falsity). Both kinds of errors can be highlighted for correction by the careful application of schema constraints. How-

[*]Corresponding author. E-mail: P.G.Omran@anu.edu.au.

ever, such constraints are commonly unavailable and, if available, uncertain and frequently violated in a KG for valid reasons, arising from the intended data-first approach of KG applications.

SHACL [6] was formally recommended by the W3C in 2017 to express such constraints on a KG as *shapes*. For example, SHACL can be used to express that a person (in a specific use case) needs to have a name, birth date, and place of birth, and that these attributes have particular types: a string; a date; a location. The shapes are used to guide the population of a KG, although they are not necessarily enforced. Typically, SHACL shapes are manually-specified. However, as for multidimensional relational database schemes [7], shapes could, in principle, be inferred from KG data. As frequent patterns, the shapes characterise a KG and can be used for subsequent data cleaning or ongoing data entry. There is scant previous research on this topic [8, 9].

While basic SHACL [6] and its advanced features [10] allows the modelling of diverse shapes including rules and constraints, most of these shapes are previously well known when expressed by alternative formalisms, including closed rules [11], trees, existential rules [12], and graph functional dependencies [13]. We claim that the common underlying form of all these shapes is the *path*, over which additional constraints induce alternative shapes. For example, in DBpedia we see the following path, $< dbo : type > \_ < db : Song > (x) \rightarrow < dbo : album > (x, y) \wedge < dbo : recordLabel > (y, z)$. which expresses that if an entity x is a song, then x is in an album y which has a record label z.

Since the satisfaction of a less-constrained shape is a necessary condition for satisfaction of a more complex shape (but not a sufficient condition), in this paper we focus on learning paths, the least constrained shape for our purposes. In addition, we learn cardinality constraints that can express, for example, that a song has at least 2 producers. We also investigate the process of constructing one kind of more complex shape, that is a *tree*, out of paths. For example, we discover a tree about an entity which has song as its type as shown in Fig. 1. In a KG context, the tree suggests that if we have an entity of type song in the KG, then we would expect to have the associated facts too.

In this paper, we present a system, SHACLEARNER, that mines shapes from KG data. For this purpose we propose a predicate calculus formalism in which rules have one body atom and a chain of conjunctive atoms in the head with a specific variable binding pattern.

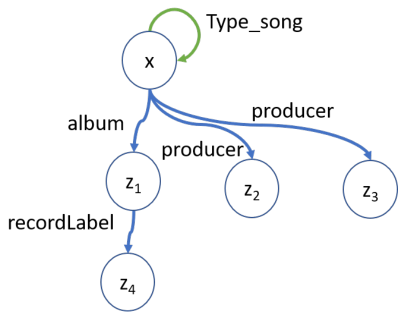

Fig. 1. A tree shape for the Song concept from DBpedia.

Since these rules are an inverse version of *open path rules* [14], we call them *inverse open path* (IOP) rules. To learn IOP rules we adapt an embedding-based open path rule learner, OPRL [14]. We define quality measures to express the validity of IOP rules in a KG. SHACLEARNER uses the mined IOP rules to subsequently discover more complex tree shapes. Each IOP rule or tree is a SHACL shape, in the sense that it can be syntactically rewritten in SHACL. Our mined shapes are augmented with a novel numerical confidence measure to express the strength of evidence in the KG for each shape.

In summary, the main contributions of this paper are:

- We introduce a new formalism called Inverse Open Path rules, that serves as a building block for more complex shapes such as trees, together with cardinality constraints and quality measurements;
- We extend the Open Path rule learning method, OPRL [14, 15], to learn IOP (**Inverse** Open Path) rules annotated with cardinality constraints, while introducing unary predicates that can act as class or type constraints; and
- We propose a method to aggregate IOP rules to produce tree shapes.

This paper is organised as follows. After presenting some foundations in Section 2, we describe our SHACL learning method in Section 3, including the formalism of IOP rules, the embedding-based method that discovers IOP rules from a KG, and the method for aggregating IOP rules into trees. In section 4, we present related work. We discuss results of an experimental evaluation in Section 5. We conclude in Section 6.

2. Preliminaries

The presentation of closed path rules and open path rules in this section is adapted and extended from [14].

An *entity e* is an identifier for an object such as a place or a person. A *fact* (also known as a *link*) is an RDF triple (e, P, e'), written here as as $P(e, e')$, meaning that the *subject* entity e is related to an *object* entity e' via the binary *predicate* (also known as a *property*), P. In addition, we admit *unary* predicates of the form $P(e)$, also equivalently written here as the binary fact $P(e, e)$. We model unary predicates as self-loops to have a unary predicate act as the label of a link in the graph, as shown in Fig. 1, just as for binary predicates. Unary predicates may, but are not limited to, represent class assertions expressed in an RDF triple as $(e, \texttt{rdf:type}, P)$ where P is a class or a datatype. A *knowledge graph* (*KG*) is a pair $K = (E, F)$, where E is a set of entities and F is a set of facts and all the entities occurring in F also occur in E.

2.1. Closed-Path Rules

KG rule learning systems employ various rule languages to express rules. RLvLR [16] and ScaleKB [17] use so-called *closed path* (CP) rules. Each consists of a *head* at the front of the implication arrow and a *body* at the tail. We say the rule is *about* the predicate of the head. The rule forms a closed path, or single unbroken loop of links between the variables. It has the following general form.

$$P_t(x, y) \leftarrow P_1(x, z_1) \wedge P_2(z_1, z_2) \wedge ... \wedge P_n(z_{n-1}, y). \tag{1}$$

We interpret these kinds of rules with universal quantification of all variables at the outside, and so we can infer a fact that instantiates the head of the rule by finding an instantiation of the body of the rule in the KG. For example, from the rule $citizenOf(x, y) \leftarrow livesIn(x, z) \wedge locatedIn(z, y)$ and the facts in the KG: $livesIn(Mary, Canberra)$ and $locatedIn(Canberra, Australia)$, we can infer and assert the new fact: $citizenOf(Mary, Australia)$.

Rules are considered of more use if they generalise well, that is, they explain many facts. To quantify this idea we recall measures *support*, *head coverage* and *standard confidence* that are used in some major approaches to rule learning including [17] and [11].

Definition 1 (satisfies, support)**.** *Let r be a CP rule of the form (1). A pair of entities (e, e')* satisfies *the body of r, denoted $body_r(e, e')$, if there exist entities $e_1, ..., e_{n-1}$ in the KG such that all of $\{P_1(e, e_1), P_2(e_1, e_2), ..., P_n(e_{n-1}, e')\}$ are facts in the KG. Further (e, e')* satisfies *the head of r, denoted $P_t(e, e')$, if $P_t(e, e')$ is a fact in the KG. Then the* support *of r counts the rule instances for which the body and the head are both satisfied in the KG.*

$$supp(r) = |\{(e, e') : body_r(e, e') \text{ and } P_t(e, e')\}|$$

Standard confidence (SC) and *head coverage* (HC) offer standardised measures for comparing rule quality. SC describes how frequently the rule is true, i.e., of the number of entity pairs that satisfy the body in the KG, what proportion of the inferred head instances are satisfied? It is closely related to *confidence* widely used in association rule mining [18]. HC measures the explanatory power of the rule, i.e., what proportion of the facts satisfying the head of the rule could be inferred by satisfying the rule body? It is closely related to *cover* which is widely used for rule learning in inductive logic programming [19]. A non-recursive rule that has both 100% SC and HC is redundant with respect to the KG, and every KG fact that is an instance of the rule head is redundant with respect to the rule.

Definition 2 (standard confidence, head coverage)**.** *Let $r, e, e', body_r$ be as given in definition 1. Then* standard confidence *is*

$$SC(r) = \frac{supp(r)}{|\{(e, e') : body_r(e, e')\}|}$$

and head coverage *is*

$$HC(r) = \frac{supp(r)}{|\{(e, e') : P_t(e, e')\}|}$$

2.2. Open-Path Rules: Rules with Open Variables

Unlike all earlier work in rule mining for KG completion, OPRL for *active* knowledge graph completion [14] defines *open path* (OP) rules of the form:

$$P_1(z_0, z_1) \wedge P_2(z_1, z_2) \wedge ... \wedge P_n(z_{n-1}, y) \rightarrow \exists x P_t(x, z_0). \tag{2}$$

Here, P_i is a predicate in the KG, each of $\{x, z_i, y\}$ are entity variables, and all free variables are universally quantified at the outside.

We call a variable *closed* if it occurs in at least two distinct predicate terms, such as z_0 here, and otherwise it is *open*. If all variables of a rule are closed then the rule is closed. An OP rule has two open variables, y and x. [14] uses OP rules since they imply the existence of a fact, like $spouse(x, y) \rightarrow \exists z\ child(x, z)$ [11]. Unlike CP rules, OP rules do not necessarily form a loop, but a straightforward variable unification transforms an OP rule to a CP rule, and every entity-instantiation of a CP rule is also an entity-instantiation of the related OP rule (but not vice-versa).

To assess the quality of OP rules, *open path standard confidence (OPSC)* and *open path head coverage (OPHC)* are derived in [14] from the closed path forms (Definition 2).

Definition 3 (open path: OPsupp, OPSC, OPHC)**.** *Let r be an OP rule of the form (2). Then a pair of entities* (e, e') satisfies *the body of r, denoted* $body_r(e, e')$*, if there exist entities* $e_1, ..., e_{n-1}$ *in the KG such that* $P_1(e, e_1)$*,* $P_2(e_1, e_2), ..., P_n(e_{n-1}, e')$ *are facts in the KG. Also* (e', e) satisfies *the head of r, denoted* $P_t(e', e)$*, if* $P_t(e', e)$ *is a fact in the KG. The* open path support, open path standard confidence*, and* open path head coverage *of r are given respectively by*

$$OPsupp(r) = |\{e : \exists e', e'' \text{ s.t. } body_r(e, e') \text{ and } P_t(e'', e)\}|$$

$$OPSC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } body_r(e, e')\}|}$$

$$OPHC(r) = \frac{OPsupp(r)}{|\{e : \exists e' \text{ s.t. } P_t(e', e)\}|}$$

2.3. SHACL Shapes

A KG is a schema-free database and does not need to be augmented with schema information natively. However, many KGs are augmented with type information that can be used to understand and validate data and can also be very helpful for inference processes on the KG. In 2017 the Shapes Constraint Language (SHACL) [6] was introduced as a W3C recommendation to define schema information for KGs stored as an RDF graph. SHACL defines constraints for graphs as *shapes*. KGs can then be validated against a set of shapes.

Shapes can serve two main purposes: validating the quality of a KG and characterising the frequent patterns in a KG. In Fig. 2, we illustrate an example of a shape from Wikidata[1], where x and z_is are variables that are instantiated by entities. Although the shape is originally expressed in ShEx [20], we translate it to SHACL here.

SHACL, together with SHACL advanced features [10] is extensive. Here we focus on the core of SHACL in which *node* shapes constrain a *target* predicate (e.g. the unary predicate *human* in Fig. 2), with *property* shapes expressing constraints over facts related to the target predicate. We particularly focus on property shapes which act to constrain an argument of the target predicate. In Fig. 2 the shape expresses that each entity x which satisfies $human(x)$ should satisfy the following paths: (1) $citizenOf(x, z_1) \wedge country(z_1)$, (2) $father(x, z_2) \wedge human(z_2)$, and (3) $nativeLanguage(x, z_3) \wedge language(z_3)$.

```
humanShape
    a sh:NodeShape;
    sh:targetClass class:human ;
    sh:property [
        sh:path property:citizenOf ;
        sh:class class:country ;
        sh:nodeKind sh:IRI ;
    ];
    sh:property [
        sh:path property:father ;
        sh:class class:human ;
        sh:nodeKind sh:IRI ;
    ];
    sh:property [
        sh:path property:nativeLanguage ;
        sh:class class:language ;
        sh:nodeKind sh:IRI ;
    ];
```

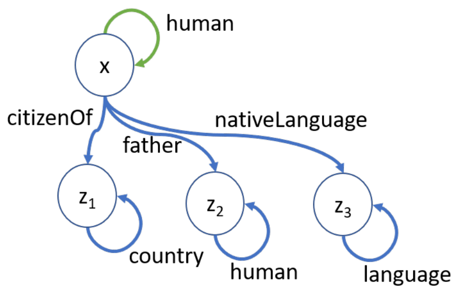

Fig. 2. An example of a SHACL shape from Wikidata.

Various formalisms with corresponding shapes have been proposed to express diverse kinds of patterns exhibited in KGs, such as k-cliques, Closed rules

[1]https://www.wikidata.org/wiki/EntitySchema:E10

(CR) [11] (that include closed path rules), Functional Graph Dependency (FGD) [13], and trees [12]. CRs are used for inferring new facts. FGDs define constraints like the type of entities in the domain and range of predicates, or the number of entities to which an entity can be related by a specific predicate. These constraints are deployed to make the KG consistent. Regardless of differences amongst these formalisms, they share a key feature in their syntax. The building block for expressing all of these shape constraints is a sequence of predicates.

We focus on such path shapes for our shape learning system. A *path* is a sequence of predicates connected by closed intermediate variables but terminating with open variables at both ends. Although shapes in the form of a path are less constrained than some more complex shapes, they are a more general template for more complex shapes like closed rules or trees, which are also paths (with further restrictions). We will define *Inverse Open Path* rules induced from paths that have a straightforward interpretation as shapes, and also propose a method to mine such rules from a KG. To demonstrate the potential for these kind of shapes to serve as building blocks for more complex shapes, we then propose a method that builds trees out of mined rules, and briefly discuss the application of such trees to KG completion.

3. SHACL learning

3.1. Rules with Open Variables or Uncertain Shapes

We observe that the converse of OP rules, that we call *inverse open path rules* (IOP), correspond to SHACL shapes. For example, the shape of Fig. 2 can be derived from the following three IOP rules:

$$human(x,x) \rightarrow citizenOf(x,z_1) \wedge country(z_1,z_1).$$

$$human(x,x) \rightarrow father(x,z_2) \wedge human(z_2,z_2).$$

$$human(x,x) \rightarrow nativeLanguage(x,z_3) \wedge language(z_3,z_3).$$

The general form of an IOP rule is given by

$$\begin{aligned} P'_t(x,z_0) \rightarrow & \exists(z_1,..z_{n-1},y)P'_1(z_0,z_1) \\ & \wedge P'_2(z_1,z_2) \wedge ... \wedge P'_n(z_{n-1},y). \end{aligned} \quad (3)$$

where each P'_i is either a predicate in the KG or its reverse with the subject and object bindings swapped, and free variables are universally quantified at the outside. We often omit the quantifiers when writing IOP rules. In an IOP rule the body of the rule is P_t and its head is the sequence of predicates, $P_1 \wedge P_2 \wedge ... \wedge P_n$. Hence we instantiate the atomic body to predict an instance of the head. IOP rules are not Horn. The pattern of existential variables in the head and universal variables in the body has been investigated in the literature as *existential rules* [12].

To assess the quality of IOP rules we follow the style of quality measures for OP rules [14].

Definition 4 (inverse open path: IOPsupp, IOPSC, IOPHC)**.** *Let r be an IOP rule of the form (3). Then a pair of entities* (e,e') satisfies *the head of r, denoted* $head_r(e,e')$*, if there exist entities* $e_1,...,e_{n-1}$ *in the KG such that* $P_1(e,e_1), P_2(e_1,e_2),...,P_n(e_{n-1},e')$ *are facts in the KG. A pair of entities* (e'',e) satisfies *the body of r, denoted* $P_t(e'',e)$*, if* $P_t(e'',e)$ *is a fact in the KG. The* inverse open path support*,* inverse open path standard confidence*, and* inverse open path head coverage *of r are given respectively by*

$$IOPsupp(r) = \mid \{e : \exists e', e'' \text{ s.t. } head_r(e,e') \text{ and } P_t(e'',e)\} \mid$$

$$IOPSC(r) = \frac{IOPsupp(r)}{\mid \{e : \exists e'' \text{ s.t. } P_t(e'',e)\} \mid}$$

$$IOPHC(r) = \frac{IOPsupp(r)}{\mid \{e : \exists e' \text{ s.t. } head_r(e,e')\} \mid}$$

Notably, because any instantiated open path in a KG induces both an OP and IOP rule: the support for an IOP rule is the same as the support for its corresponding OP form; IOPSC is the same as the corresponding OPHC; and IOPHC is the same as the corresponding OPSC. This close relationship between OP and IOP rules helps us to mine both OP and IOP rules in the one process.

We show the relationship between an OP rule and its converse IOP version in the following example. Consider the OP rule, $P_1(x,z_0) \leftarrow P_2(z_0,z_1) \wedge P_3(z_1,z_2)$. Assume we have three entities ($\{e_3,e_4,e_5\}$) which can instantiate z_0 and satisfy both $P_1(x,z_0)$ and $P_2(z_0,z_1) \wedge P_3(z_1,z_2)$. Assume the number of entities that can instantiate z_0 to satisfy the head part is 5 ($\{e_1,e_2,e_3,e_4,e_5\}$) and the number of entities that can instantiate z_0 to satisfy the body part is 7 ($\{e_3,e_4,e_5,e_6,e_7,e_8,e_9\}$). Hence, we have the following for this rule, $OPsupp = 3$, $OPSC = 3/7$ and $OPHC = 3/5$. For the IOP version of the rule over the same KG, $P_1(x, z_0) \rightarrow P_2(z_0, z_1) \wedge P_3(z_1, z_2)$, we have the same entities to instantiate z_o while the predicate terms corresponding to the bodies and heads are swapped. Hence, we have the following for the IOP version of the rule, $IOPsupp = 3$, $IOPSC = 3/5$ and $IOPHC = 3/7$.

In many cases we need the rules to express not only the necessity of a chain of facts (the facts in the head of the IOP rule) but the number of different chains which should exist. For example, we may need a rule to express that each human has at least two parents. Hence, we introduce IOP rules annotated with a cardinality, *Car*. This gives the following annotated form for each IOP rule.

$$IOPSC, IOPHC, Car : P'_t(x, z_0) \rightarrow P'_1(z_0, z_1) \wedge P'_2(z_1, z_2) \wedge ... \wedge P'_n(z_{n-1}, y). \quad (4)$$

where IOPSC and IOPHC belong to $[0, 1]$ and denote those qualities of the rule. *Car* is an integer $\geqslant 1$.

Definition 5 (Cardinality of an IOP rule, Car)**.** *Let r be an annotated IOP rule of the form (4) and let $Car(r)$ be the cardinality annotation for r. Then r* satisfies *$Car(r)$ iff for each entity $e \in \{e | \exists e'' s.t.\ P_t(e'', e)\}$, $Car(r) \leqslant \mid \{e' | head_r(e, e'\} \mid$.*

The cardinality expresses a lower bound on the number of head paths which are satisfied in the KG for every instantiation of the variable that joins the body to the head. For example, if we have $0.8, 0.1, 2 : P_t(x, y) \rightarrow P_1(y, z) \wedge P_2(z, t)$ and have $P_t(e_1, e_2)$ satisfied, we should have at least two paths starting from e_2 that instantiate two distinct entities for variable *t*, of the form $P_1(e_2, z) \wedge P_2(z, e_3)$ and $P_1(e_2, z) \wedge P_2(z, e_4)$. There is no limitation on the instantiations for variable *z* which is scoped inside the head, so these two pairs of instantiations both satisfy cardinality of 2: (1) $P_1(e_2, e_5) \wedge P_2(e_5, e_3)$ and $P_1(e_2, e_5) \wedge P_2(e_5, e_4)$ and (2) $P_1(e_2, e_5) \wedge P_2(e_5, e_3)$ and $P_1(e_2, e_6) \wedge P_2(e_6, e_4)$.

Rules with the same head and the same body may have different cardinalities. In the given example we might have the following rule as well, $0.9, 0.1, 1 : P_t(x, y) \rightarrow P_1(y, z) \wedge P_2(z, t)$. While we have a rule with a cardinality, c_1, we also have rules with lower cardinalities $1, ..., (c_1 - 1)$ but their IOPSCs should be as good or better than the rule with c_1 cardinality. This statement is quite intuitive since cardinality expresses a lower bound on the number of instances of the head in the KB.

Lemma 1 (IOPSC is non-increasing with length)**.** *Let r be an IOP rule of the form (3) with $n \geqslant 2$ and let r' be an IOP rule of the form $P'_t(x, z_0) \rightarrow P'_1(z_0, z_1) \wedge P'_2(z_1, z_2) \wedge ... \wedge P'_n(z_{n-2}, y)$, being r shortened by the removal of the last head predicate. Then $IOPSC(r) \leqslant IOPSC(r')$.*

Proof. Observe that by definition 4, the denominator of $IOPSC()$ is not affected by the head of the rule, and so has the same value for both. Now, looking at the numerators, we have that
$\forall e(\exists e_1, .., e_n(P'_1(e, e_1) \wedge P'_2(e_1, e_2) \wedge ... \wedge P'_n(e_{n-2}, e_{n-1}) \wedge P'_n(e_{n-1}, e_n)) \implies \exists e_1, .., e_{n-1}(P'_1(e, e_1) \wedge P'_2(e_1, e_2) \wedge ... \wedge P'_n(e_{n-2}, e_{n-1})))$. Therefore
$\{e : \exists e'\ s.t.\ head_r(e, e')\} \subseteq \{e : \exists e'\ s.t.\ head_{r'}(e, e')\}$.
Therefore $\{e : \exists e', e''\ s.t.\ head_r(e, e')\ and\ P_t(e'', e)\} \subseteq \{e : \exists e', e''\ s.t.\ head_{r'}(e, e') and\ P_t(e'', e)\}$.
So $IOPsupp(r) \leqslant IOPsupp(r')$ as required. □

3.2. IOP Learning through Representation Learning

To mine IOP rules, we start with the open path rule learner OPRL [14], and adapt its embedding-based OP rule learning to learn annotated IOP rules. We call this new IOP rule learner, SHACLEARNER, shown in Algorithm 1.

SHACLEARNER (Algorithm 1) uses a sampling method Sampling(), Algorithm 2, to prune the entities and predicates that are less relevant to the target predicate to obtain a sampled KG. The sample is fed to an embedding learner, RESCAL [21, 22], in Embeddings(). Then in PathFinding(), SHACLEARNER uses the computed embedding representations of predicates and entities in heuristic functions that inform the generation of IOP rules bounded by a maximum length. Then, potential IOP rules are evaluated, annotated, and filtered in Ev() to produce annotated IOP rules. Eventually, a tree is discovered for each argument of each target predicate by aggregating mined IOP rules in GreedySearch().

While the overall algorithm structure of SHACLEARNER is similar to OPRL [14], as is the embedding-based scoring function, the following elements are novel in SHACLEARNER:

- OPRL cannot handle unary predicates while SHACLEARNER admits unary predicates both in the head and the body of IOP rules.
- SHACLEARNER can discover and evaluate IOP rules with minimum cardinality constraints in the head of the IOP rule, while OPRL is effectively

Algorithm 1 SHACLEARNER

Input: a KG K, a target predicate P_t
Parameters: max rule length l, max rule cardinality $MCar$, $MinIOPSC$, $MinIOPHC$, and $MinTreeSC$
Output: a set of IOP rules R and $Tree$

```
1: $K' :=$ Sampling$(K, P_t)$
2: $(\mathcal{P}, \mathcal{A}) :=$ Embeddings$(K')$
3: $R' := \emptyset$
4: for $2 \leqslant k \leqslant l$ do
5:     Add PathFinding$(K', P_t, \mathcal{P}, \mathcal{A}, k)$ to $R'$
6: end for
7: $R :=$ Ev$(R', K, MCar, MinIOPSC, MinIOPHC)$
8: $Tree :=$ GreedySearch$(R, MinTreeSC)$
9: return $Tree$ and $R$
```

limited to learning the special case of minimum cardinality 1 for all rules. For this reason, the evaluation method of SHACLEARNER, Ev(), differs from the OPRL evaluation module.

– The aggregation module that produces trees out of learnt IOP rules, ready for translation to SHACL, is novel in SHACLEARNER.

3.3. Sampling

In more detail, the Sampling() method in line 1 of Algorithm 1 computes a fragment of the KG (K') consisting of a bounded number of entities that are related to the goal predicate (i.e., P_t). This sampling is essential since embedding learners (e.g., HOLE [21] and RESCAL) cannot handle massive KGs with millions of entities (e.g. YAGO2).

The sampling method, first introduced in [16], is shown in Algorithm 2. Since we search for IOP rules with up to l atoms (including the specific body target predicate, P_t), the entity set E_{sample} and corresponding fact set K' contains the information needed for learning such rules. Predicates less relevant to the target predicate are pruned in the sampling process and no facts about those predicates remain in K'.

3.4. Embeddings

After sampling, in line 2 Embeddings(), we compute predicate embeddings as well as subject and object argument embeddings for all predicates in the sampled K', as is done in RLvLR [16]. The embedding is obtained from RESCAL [21, 22] as it can provide an extensive representation of predicates and entities as is shown in previous heuristic rule learners like [23].

Algorithm 2 Sampling

Input: a KG K, a target predicate P_t
Parameters: max rule length l
Output: K' a subgraph of the input graph

```
1: $E_1 = \{e \mid \exists e' : P_t(e, e') \vee P_t(e', e)\}$
2: for $2 \leqslant k \leqslant l$ do
3:     $E_k = \{e \mid \exists e' : (e' \in E_{k-1}) \wedge (P_i(e, e') \vee P_i(e', e))\}$
4: end for
5: $E_{sample} = \bigcup_1^l E_i$
6: $K' := \{P_i(e, e') \mid (e \in E_{sample}) \wedge e' \in (E_{sample})\}$
7: return $K'$
```

Briefly, we use RESCAL [21] to embed each entity e_i to a vector $\mathbf{E}_i \in \mathbb{R}^d$ and each predicate P_k to a matrix $\mathbf{P}_k \in \mathbb{R}^{d \times d}$ where $\mathbb{R}$ is the set of real numbers and d is an integer (a parameter of RESCAL). For each given fact $P_0(e_1, e_2)$, the following scoring function is computed:

$$f(e_1, P_0, e_2) = \mathbf{E}_1^T \cdot \mathbf{P}_0 \cdot \mathbf{E}_2 \tag{5}$$

The scoring function indicates the plausibility of the fact that e_1 has relation P_0 with e_2.

In Embeddings() we additionally compute argument embeddings according to the method proposed in [16]. To compute the subject (respectively object) argument embeddings of a predicate P_k, we aggregate the embeddings of entities that occur as the subject (respectively object) of P_k in the KG. Hence for each predicate P_k we have two vectors, $\mathbf{P}_k^1$ and $\mathbf{P}_k^2$ that present the subject argument and object argument of P_k respectively.

3.5. Generating and Pruning Rules

After that, in line 3 to line 7 of Algorithm 1, PathFinding() produces candidate IOP rules based on the embedding representation of the predicates which are involved in each rule. The candidate rules are pruned by the scoring function heuristic proposed in [14] for OP rules. Due to the close relationship between OP and IOP rules, a high-scoring candidate suggests both a good OP rule and a good IOP rule.

An IOP rule $P_t(x, y) \to P_1(y, z) \wedge P_2(z, t)$. acts to connect entities satisfying the subject argument of the body predicate, P_t, to entities forming the object argument of the last predicate, P_2, along a path of entities that satisfy a chain of predicates in the rule. There is a relationship between the logical statement of the rule and the following properties in the embedding space [14]:

1. The predicate arguments that have the same variable in the rule should have similar argument emdeddings. For example we should have the following similarities, $\mathbf{P}_t^2 \sim \mathbf{P}_1^1$ and $\mathbf{P}_1^2 \sim \mathbf{P}_2^1$.
2. The whole path forms a composite predicate, like $P^*(x,t) = P_t(x,y) \wedge P_1(y,z) \wedge P_2(z,t)$. We compute the embedding representation of the composite predicate based on its components, $\mathbf{P}^*(x,t) = \mathbf{P}_t(x,y) \cdot \mathbf{P}_1(y,z) \cdot \mathbf{P}_2(z,t)$. Now we could check the plausibility of $P^*(x,t)$ for any pair of entities by equation 5. However, since we are interested in the existence of an entity-free rule, the following similarity will hold: $1 \approx \mathbf{P}_t^1 \cdot \mathbf{P}_t \cdot \mathbf{P}_1 \cdot \mathbf{P}_2 \cdot \mathbf{P}_2^2$.

Based on the above two properties, [14] defines two scoring functions that help us to heuristically mine the space of all possible IOP rules to produce a reduced set of candidate IOP rules.

The ultimate evaluation of an IOP rule will be done in the next step as described below.

3.6. *Efficient Computation of Quality Measures*

Now we focus our attention on the efficient matrix-computation of the quality measures that are novel for SHACLEARNER. Ev() in Algorithm 1 first evaluates candidate rules on the small sampled KG, and selects only the rules with $IOPsupp(r) \geqslant 1$. They may still include a large number of redundant and low quality rules and so are further downselected based on their IOPSC and IOPHC calculated over the full KG. We show in the following how to efficiently compute the measures over massive KGs using an *adjacency matrix* representation of the KG.

Let $K = (E, F)$ with $E = \{e_1, \ldots, e_n\}$ be the set of all entities and $\mathbb{P} = \{P_1, \ldots, P_m\}$ be the set of all predicates in F. Like RESCAL [21, 22], we represent K as a set of square $n \times n$ adjacency matrices by defining the function $\mathcal{A}$. Specifically, the $[i, j]$th element $\mathcal{A}(P_k)[i, j]$ is 1 if the fact $P_k(e_i, e_j)$ is in F; and 0 otherwise. Thus, $\mathcal{A}(P_k)$ is a matrix of binary values and the set $\{\mathcal{A}(P_k) \mid k \in \{1, \ldots, m\}\}$ represents K.

We illustrate the computation of IOPSC and IOPHC through an example. Consider the IOP rule r : $P_t(x, z_0) \to P_1(z_0, z_1) \wedge P_2(z_1, y)$. Let $E = \{e_1, e_2, e_3\}$ and

$$F = \{P_1(e_1,e_2), P_1(e_2,e_1), P_1(e_2,e_3), P_1(e_3,e_1), P_2(e_1,e_2), P_2(e_3,e_2), P_2(e_3,e_3), P_t(e_1,e_3), P_t(e_3,e_2), P_t(e_3,e_3)\}$$

The adjacency matrices for the predicates P_1, P_2 and P_t are:

$$\mathcal{A}(P_1) : \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \mathcal{A}(P_2) : \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix},$$

$$\mathcal{A}(P_t) : \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

For IOPSC and IOPHC (definition 4) we need to calculate (1) the number of entities that satisfy the body of the rule, i.e.,
$\#e : \exists e''$ *s.t.* $P_t(e'', e)$, (2) the number of entities that satisfy the head of a rule i.e., $\#e : \exists e'$ *s.t.* $head_r(e, e')$ and, (3) the number of entities that join the head of a rule to its body i.e.,
$\#e : \exists e', e''$ *s.t.* $head_r(e, e')$ *and* $P_t(e'', e)$.

For (1) we can read the pairs (e'', e) directly from the matrix $\mathcal{A}(P_t)$. To find distinct es we sum each *column* (corresponding to each value for the second argument) and transpose to obtain the vector $\mathcal{V}^2(P_t)$. Each non-zero element of this vector indicates a satisfying e and the number of distinct es is given by simply counting the number of non-zero elements. In the example, the only non-zero element in $\mathcal{A}(P_t)$ is $\mathcal{A}(P_t)[1, 3]$ and after summing the columns and transposing we have

$$\mathcal{V}^2(P_t) = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$$

so we have only $\mathcal{V}^2(P_t)[3]$ non-zero and $\{e_2, e_3\}$ satisfies the head with count 2.

For (2) the pairs (e, e') satisfying the head are connected by the path $P_1, P_2, \ldots P_m$, and can be obtained, as for (1), directly from the matrix product $\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m)$, being the elements with a value $\geqslant$ *Car* (for rules with cardinality *Car*). To find distinct es we sum each *row* (corresponding to each value for the first argument) to obtain the vector $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))$. Each element with

$\geqslant Car$ value of this vector indicates a satisfying e and the number of distinct es is given by counting the number of elements in $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))$.

In the example we have

$$\mathcal{A}(P_1)\mathcal{A}(P_2) = \begin{bmatrix}0 & 0 & 0\\ 0 & 2 & 1\\ 0 & 1 & 0\end{bmatrix}, \mathcal{V}^1(P_1, P_2) = \begin{bmatrix}0\\ 3\\ 1\end{bmatrix}$$

with satisfying entities e_2 and e_3 and count of 2 for $Car = 1$. For $Car = 2$, e_2 satisfies the rule so the count is 1.

Computing (3) is now straightforward. We have that the row index of non-zero elements of $\mathcal{V}^2(P_t)$ indicates entities that satisfy the second argument of the body and that row index of elements with a value $\geqslant$ Car of $\mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))$ indicates entities that satisfy the first argument of the head. Therefore we can find the entities that satisfy both of these conditions by pairwise multiplication. That is, the entities we need are $\{e_i \mid (\mathcal{V}^2(P_t)[i] > 0 \wedge \mathcal{V}^1(\mathcal{A}(P_1)\mathcal{A}(P_2)\ldots\mathcal{A}(P_m))[i]) \geqslant Car \text{ and } 1 \leqslant i \leqslant n\}$, and the count of entities is the size of this set. For the example, for $Car = 1$, we have $\{e_2, e_3\}$ in the set with count 2 and for $Car = 2$, we have $\{e_2\}$ in the set with count 1.

Hence, we could have three versions of r, i.e., r^1, r^2 and r^3 with three different Car of 1, 2, and 3 respectively. For $Car = 1$, $IOPsupp(r^1) = |\{e_2, e_3\}| = 2$. From Definition 4 we can obtain $IOPHC(r^1) = 2/2$ and $IOPSC(r^1) = 2/2$. For $Car = 2$, $IOPsupp(r^2) = |\{e_2\}| = 1$. We can obtain $IOPHC(r^2) = 1/1$ and $IOPSC(r^2) = 1/2$. In this case, we have the same qualities for $Car = 3$ as we have for $Car = 2$.

3.7. From IOP Rules to Tree Shapes

Now in line 8 of Algorithm 1 we turn to deriving SHACL trees, as illustrated in Figure 3, from annotated IOP rules. This procedure is novel in SHACLEARNER. We use a greedy search to aggregate the IOP rules for the subject argument and the object argument of each target predicate.

For example, the shape of Fig. 2 has the following tree:

$$\begin{aligned} human(x,x) \rightarrow\ & citizenOf(x,z_1) \wedge country(z_1,z_1)\\ & \wedge father(x,z_2) \wedge human(z_2,z_2)\\ & \wedge nativeLanguage(x,z_3)\\ & \wedge language(z_3,z_3). \end{aligned}$$

The general form of a tree is given by

$$\begin{aligned} & P'_t(x,z_0) \rightarrow \exists(z^*_*, y^*)\\ & \quad P'^1_1(z_0,z^1_1) \wedge P'^1_2(z^1_1,z^1_2) \wedge \ldots \wedge P'^1_n(z^1_{n-1},y^1)\\ & \quad \wedge P'^2_1(z_0,z^2_1) \wedge P'^2_2(z^2_1,z^2_2) \wedge \ldots \wedge P'^2_m(z^2_{m-1},y^2)\\ & \quad \ldots\\ & \quad \wedge P'^q_1(z_0,z^q_1) \wedge P'^q_2(z^q_1,z^q_2) \wedge \ldots \wedge P'^q_t(z^q_{t-1},y^q) \end{aligned} \tag{6}$$

where each P'^j_i is either a predicate in the KG or its reverse with the subject and object bindings swapped. Free variables are universally quantified at the outside. In a tree we say the body of the shape is P_t and its head is the sequence of paths or branches, $Path^1 \wedge Path^2 \wedge \ldots \wedge Path^q$. Hence we instantiate the atomic body to predict an instance of the head. All head branches and the body join in one shared variable, z_0.

To assess the quality of a tree we follow the quality measures for IOP rules.

Definition 6 (Tree: Treesupp, TreeSC). *Let r be a tree of the above form. Then a set of pairs of entities $(e,e^1),\ldots,(e,e^q)$* satisfies *the head of r, denoted $head_r(e)$, if there exist the following sequences of entities $e^1_1,\ldots,e^1_{n-1}$, $e^2_1,\ldots,e^2_{m-1}$ and $e^q_1,\ldots,e^q_{t-1}$ in the KG such that $P^1_1(e,e^1_1), P^1_2(e^1_1,e^1_2),\ldots,P^1_n(e^1_{n-1},e^1)$, $P^2_1(e,e^2_1), P^2_2(e^2_1,e^2_2)$, $\ldots, P^2_m(e^2_{m-1},e^2)$ and $P^q_1(e,e^q_1)$, $P^q_2(e^q_1,e^q_2)$ $,\ldots,P^q_t(e^q_{t-1},e^q)$ are facts in the KG. A pair of entities (e'',e)* satisfies *the body of r, denoted $P_t(e'',e)$, if $P_t(e'',e)$ is a fact in the KG. The* tree support *and* tree standard confidence *of r are given respectively by*

$$Treesupp(r) = \mid \{e : \exists e'' \text{ s.t. } head_r(e) \text{ and } P_t(e'',e)\} \mid$$

$$TreeSC(r) = \frac{Treesupp(r)}{\mid \{e : \exists e'' \text{ s.t. } P_t(e'',e)\} \mid}$$

To learn each tree we employ a greedy search, GreedySearch, in line 8 of Algorithm 1. To do so, we sort all rules that bind the subject argument (for the left hand tree in Fig. 3) in a non-increasing order with respect to IOPSC Then we iteratively try to add the first rule in the list to the tree and compute the TreeSC. If the TreeSC drops below the defined threshold (i.e.,

$TreeSCMIN$) we dismiss the rule otherwise we add it to the tree. For the right hand tree we do the same with the rules that bind the object argument of the target predicate. Since a conjunction of IOP rules form a tree, TreeSC is bounded above by the minimum IOPSC of its constituent IOP rules.

Aside from the cardinality, the tree may be straightforwardly interpreted as a set of SHACL shapes by reading off every path from the target predicate terminating at a node in the tree. The body predicate is declared a `sh:nodeShape` and the path of head predicates as nested `sh:path` declarations within a `sh:property` declaration. Cardinality of a path is read from the annotation of the branch at the terminating node, and declared by `sh:minCount` within the property declaration. See section 3.1 for an example.

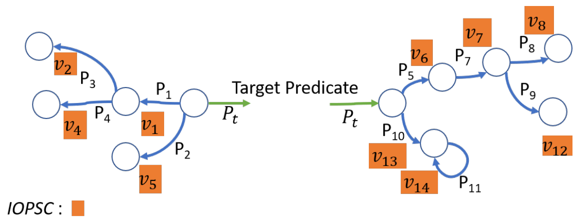

Fig. 3. Trees for a target predicate P_t. Each v_i indicates a TreeSC.

3.7.1. Tree Shapes are Useful for Human Interaction

Shapes offer KG documentation as readable patterns and also provide a way to validate a KG. Our novel tree shapes can additionally be used for KG-completion. While there are several methods proposed to complete KGs automatically (e.g., [16, 21, 22]) by predicting missing facts (also called *links*), all these methods traverse the KG in a breadth-first manner. In other words, they sequentially answer a set of independent questions such as $birthPlace(Trump, ?)$ followed by $playsIn(Naomi\_Watts, ?)$. Our proposed tree shapes instead provide an opportunity to work sequentially along a path of dependent questions such as $birthPlace(Trump, ?_1)$ followed by $capitalOf(?_1, ?)$. The latter question cannot even be asked until we have an answer for the former question, and the existence of an answer to the former gives us the confidence to proceed to the next question along the path. This completion strategy is *depth-first* as it works through a shape tree. Importantly, when we want to ask such completion questions of a human, this depth-first questioning strategy will reduce the cognitive load due to the contextual connections between successive questions. This strategy for human-KG-completion is applied in the smart KG editor *Schímatos* [24] using trees that can be generated by SHACLEARNER.

Tree shapes can also help a human expert to extract a more intuitive concise sub-tree out of a deeper, more complex tree when desired for human interpretation. If a tree with confidence $TreeSC_{orig}$ is pruned either by removing branches (width-wise) or by reducing the length of branches (depth-wise), it remains a valid tree with confidence $TreeSC_{new}$ with the property that $TreeSC_{new} \geqslant TreeSC_{orig}$. Hence, by pruning a tree we obtain a simpler tree with higher confidence in the KG.

4. Related Work

There are some exploratory attempts to address learning SHACL shapes from KGs [9, 25–28]. They are procedural methods without logical foundations and are not shown to be scalable to handle real-world KGs. They work with a small amount of data and the representation formalism they use for their output is difficult to compare with the IOP rules which we use in this paper. [28] carries out the task in a semi-automatic manner: it provides a sample of data to an off-the-shelf graph structure learner and provides the output in an interactive interface for a human user to create SHACL shapes.

In [8] the authors provide an interactive framework to define SHACL shapes with different complexities, including nested patterns that are similar to the trees that we use. A formalisation of the approach is given, but there are no shape quality measures that are essential for large scale shape mining. Because the paper does not provide a system that discovers patterns from massive KGs, we can not deploy their method for comparison purposes.

There are some works [29, 30] that use existing ontologies for KGs to generate SHACL shapes. [29] uses two different kinds of knowledge to automatically generate SHACL shapes: ontology constraint patterns as well as input ontologies. In our work we use the KG itself to discover the shapes, without relying on external modelling artefacts.

From an application point of view, there are papers which investigate the application of SHACL shapes to the validation of RDF databases including [31, 32], but these do not contribute to the discovery of shapes.

[33] proposes an extended validation framework for the interaction between inference rules and SHACL shapes in KGs. When a set of rules and shapes are pro-

vided, a method is proposed to detect which shapes could be violated by applying a rule.

There are some attempts to provide logical foundations for the semantics of the SHACL language including [34] that presents the semantics of recursive SHACL shapes. By contrast, in our work we approach SHACL semantics in the reverse direction. We start with logical formalisms with both well-defined semantics and motivating use cases to derive shapes that can be trivially expressed in a fragment of SHACL.

5. Experiments

We have implemented our SHACLEARNER[2] based on Algorithm 1, and conducted experiments to assess it. Our experiments are designed to prove the effectiveness of our SHACLEARNER at capturing shapes with varying confidence, length and cardinality from various real-world massive KGs. Since our proposed system is the first method to learn shapes from massive KGs automatically, we have no benchmark with which to compare. However, the performance of our system shows that it can handle the task satisfactorily and can be applied in practice. We demonstrate that:

1. SHACLEARNER is scalable so it can handle real-world massive KGs including DBpedia with over 11M facts.
2. SHACLEARNER can learn several shapes each for various target predicates.
3. SHACLEARNER can discover diverse shapes with respect to the quality measurements of IOPSC and IOPHC.
4. SHACLEARNER discovers shapes of varying complexity, and diverse with respect to length and cardinality.
5. SHACLEARNER discovers *every* high-quality rule (with $IOPSC \geqslant 0.9$) for a small complete KG, by comparison with an ideal learner.
6. SHACLEARNER discovers more complex shapes (trees) by aggregating learnt IOP rules efficiently.

Our four benchmark KGs are described in Table 1. Three benchmarks, namely YAGO2s, Wikidata, and DBpedia, are common KGs and have been used in rule learning experiments previously [11, 16]. The fourth is a small synthetic KG, Poker, for analysing the completeness of our algorithm. The Poker KG was introduced in [14], adapted from the classic version [35, 36] to be a rich and correct KG for evaluation experiments. Each poker hand comprises 5 playing cards drawn from a deck with 13 ranks and 4 suits. Each card is described using two attributes: suit and rank. Each hand is assigned to any or all of 9 different ranks, including High Card, One Pair, Two Pair, etc. We randomly generate 500 poker hands and all facts related to them to build a small but complete and correct KG as summarised in Table 1. 28 out of 35 predicates are unary predicates, such as fullHouse(x) where x is a specific poker hand.

[2]Detailed experimental results can be found at https://www.dropbox.com/sh/dn1kujeg0b6o9bw/AAAbKG9KfZ7e0eaNONz2zE93a?dl=0

Table 1
Benchmark specifications

KG	# Facts	# Entities	# Predicates
YAGO2s	4,122,438	2,260,672	37
Wikidata	8,397,936	3,085,248	430
Wikidata +UP	8,780,716	3,085,248	587
DBpedia 3.8	11,024,066	3,102,999	650
DBpedia 3.8 +UP	11,498,575	3,102,999	1,005
Poker	6,790	575	35

All experiments were conducted on an Intel Xeon CPU E5-2650 v4 @2.20GHz, 66GB RAM and running CentOS 8.

5.1. Transforming KGs with type predicates for experiments

Since real-world KG models treat predicates and entities in a variety of ways, we require a common representation for this work that clearly distinguishes entities and predicates. We employ an abstract model that is used in Description Logic ontologies, where classes and types are named unary predicates, and roles (also called properties) are named binary predicates.

In real-world KGs, concept or class membership may be modeled as entity instances of a binary fact. For example, DBpedia contains "<dbo:type>(x, y)" and "<dbo:class>(x, y)" predicates where the second arguments of these predicates are types (e.g. "<db:Album>" or "<db:City>") or classes (e.g. "<db:Reptile>" or "<db:Bird>"). Instead, we choose to model types and classes with *unary* predicates. To do so we make new predicates like <dbo:type>_<db:Album>(x) from facts in the form <dbo:type>$(x$,<db:Album>), where x is the name of an album.

Then we produce new unary facts based on the new predicate and related facts. For example, for <dbo:type>(Revolver,<db:Album>), we produce the new fact <dbo:type>_<db:Album>(Revolver).

We manually choose two predicates from DBpedia 3.8 "<dbo:type>" and "<dbo:class>" and one from Wikidata "<occupation_P106>" to generate our unary predicates and facts. These predicates each have a class as their second argument. To prune the classes with few instances for which learning may be pointless, we consider only our unary predicates which have at least 100 facts. We retain the original predicates and facts in the KG as well as extending it with our new ones. In Table 1 we report the specifications of two benchmarks where we have added the unary predicates and facts (denoted as +UP).

5.2. Learning IOP Rules

We follow the established approach for evaluating KG rule-learning methods, that is, measuring the quantity and quality of distinct rules learnt. Rule quality is measured by Inverse open path standard confidence (IOPSC) and Inverse open path head coverage (IOPHC). We randomly select 50 target predicates from Wikidata and DBPedia unary predicates (157 and 355 respectively). We use all binary predicates of YAGO2s (i.e., 37) as target predicates. Each binary target predicate serves as two target predicates, once in the straight form (where the object argument of the predicate is the common variable to connect the head) and secondly in its reverse form (where the subject argument serves to connect). In this manner, we ensure that the results of SHACLEARNER on YAGO2s with its binary predicates as targets is comparable with the results for Wikidata and DBpedia that have unary predicates as targets. Hence for YAGO2s we have 74 target predicates.

A 10 hour limit is set for learning each target predicate.

Table 2 shows #Rules, the average numbers of quality IOP rules found; %SucTarget, the proportion of target predicates for which at least one IOP rule was found; and the running times in hours, averaged over the targets. Only high quality rules meeting minimum quality thresholds are included in these figures, that is, with IOPSC$\geqslant$ 0.1 and IOPHC$\geqslant$ 0.01, thresholds established in comparative work [11].

SHACLEARNER shows satisfactory performance in terms of both the runtime and the numbers of quality rules mined. Note that rules found have a variety of lengths and cardinalities. To better present the quality performance of rules we illustrate the distribution of rules with respect to the features, IOPSC, IOPHC, cardinality and length, and also IOPSC vs length. In the following, the *proportion* of mined rules having the various feature values is presented, to more evenly demonstrate the quality of performance over the three very different KGs.

Table 2
Performance of SHACLEARNER on benchmark KGs

Benchmark	%SucTarget	#Rules	Time (hours)
YAGO2s	80	42	0.6
Wikidata+UP	82	67	1.8
DBpedia+UP	98	157	2.4

The distribution of mined rules with respect to their IOPSC and IOPHC is shown in Figure 4. In the left-hand chart we observe a consistent decrease in the proportion of quality rules as the IOPSC increases. In the right hand chart we see a similar pattern for increasing IOPHC, but the decrease is not as consistent, showing there must be many relevant rules with many covered head instances. Over all benchmarks, the majority of learned rules have lower IOPSC and IOPHC. This is expected because the statistical likelihood of poor quality rules is much higher. Indeed, we show experimentally in section 5.3 that our pruning techniques, that are necessary for scalability, prune away predominantly lower quality rules.

With respect to IOPSC, proportionally more quality rules are learnt from DBpedia than from the other KGs, with Wikidata second, ahead of YAGO2s. This phenomenon might be caused by the way that we select the target predicates: for Wikidata and DBpedia we handpick unary predicates that represent classes, while for YAGO2 we consider all binary predicates.

The distribution of mined rules with respect to their cardinalities is shown in Figure 5. We observe that the largest proportion of rules has a cardinality of 1, as expected, as they have the least stringent requirements to be met in the KG. We observe an expected decrease with greater cardinalities as they demand tighter restrictions to be satisfied. YAGO2 demonstrates a tendency towards higher cardinalities than the other KGs, possibly a result of its more curated development.

The distribution of mined rules with respect to their lengths is shown in Figure 6. One might expect that as the length increases, the number of rules would increase since the space of possible rules grows, and this is what we see.

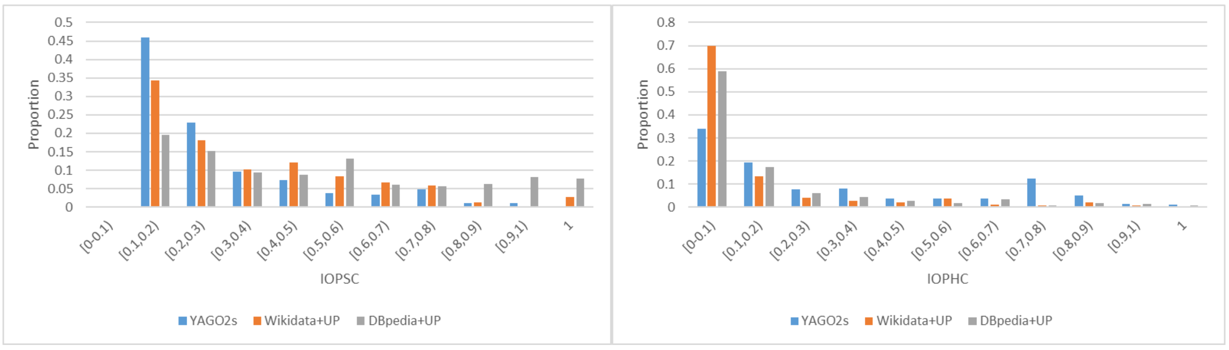

Fig. 4. Distribution of mined rules with respect to IOPSC and IOPHC

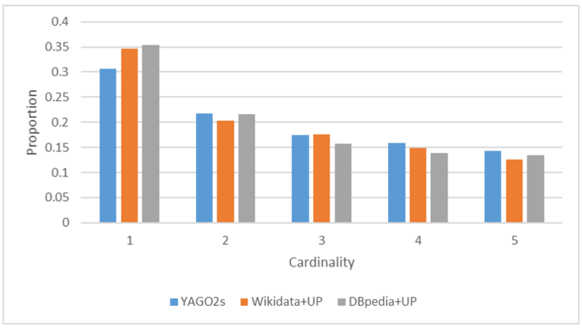

Fig. 5. Distribution of mined rules with respect to their cardinalities

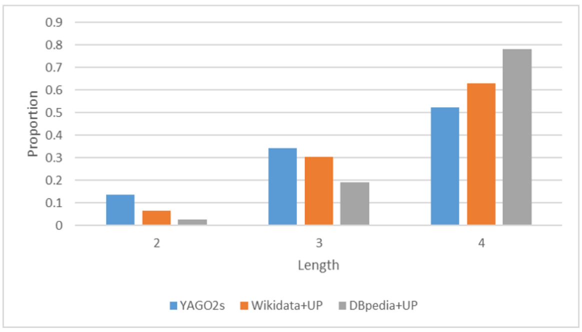

Fig. 6. Distribution of mined rules with respect to their lengths

For a concrete example of SHACL learning, we show the following three IOP rules mined from DBpedia in the experiments. The IOPSC, IOPHC, and Cardinality annotations respectively prefix each rule.

$$0.64, 0.01, 1 : <\text{dbo}:\text{type}>\_<\text{db}:\text{Song}>(x) \rightarrow <\text{dbo}:\text{album}>(x, z_1) \wedge <\text{dbo}:\text{recordLabel}>(z_1, y).$$

$$0.63, 0.01, 1 : <\text{dbo}:\text{type}>\_<\text{db}:\text{Song}>(x) \rightarrow <\text{dbo}:\text{producer}>(x, y).$$

$$0.41, 0.01, 2 : <\text{dbo}:\text{type}>\_<\text{db}:\text{Song}>(x) \rightarrow <\text{dbo}:\text{producer}>(x, y).$$

$<\text{dbo}:\text{type}>\_<\text{db}:\text{Song}>(x)$ expresses x is a song. The first rule indicates x should belong to an album (z_1) that has y as record label. The second rule requires a song (x) to have at least one producer while the third rule requires a song to have at least two producers, and these two rules are distinguished by the cardinality annotation. As we discussed in 3.1, the third rule is more constraining than the second, so the confidence of the third rule is lower than the confidence of the second, based on the KG data.

The rules can be trivially rewritten (through a script that we developed) into SHACL syntax as follows.

```
:s1 a sh:NodeShape ;
    sh:targetClass :<dbo:type>_<db:Song> ;
    sh:property [
        sh:minCount 1 ;
        sh:path :<dbo:album> ;
            [sh:path :<dbo:recordLabel> ; ]
    ] .
:s2 a sh:NodeShape ;
    sh:targetClass :<dbo:type>_<db:Song> ;
    sh:property [
        sh:minCount 1 ;
        sh:path :<dbo:producer> ;
    ] .
:s3 a sh:NodeShape ;
    sh:targetClass :<dbo:type>_<db:Song> ;
    sh:property [
        sh:minCount 2 ;
        sh:path :<dbo:producer> ;
    ] .
```

5.2.1. *IOPSC vs IOPHC*

Using rules found in the experiments, we further illustrate the practical meaning of the IOPSC and IOHC qualities. While IOPSC determines the confidence of a rule based on counting the proportion of target predicate instances for which the rule holds true in the KG, IOPHC indicates the proportion of rule consequent instances that are justified by target predicate instances in the KG, thereby indicating the relevance of the rule to the target. In Wikidata+UP, all unary predicates are occupations such as singer or entrepreneur, so all the entities which have these types turn out to be persons even though there is no explicit person type in our KG. Thus, the occupations all have very similar IOP rules about each of them with high IOPSC and low IOPHC, like the following one, for example.

$$0.47, 0.01, 1 : < \text{occupation} > \_ < \text{singer} > (x) \rightarrow < \text{country\_of\_citizenship} > (x, y).$$

On the other hand, for these unary occupation predicates there are also some IOP rules with high IOPHC that apply only to one specific unary predicate, such as the following one.

$$0.15, 0.16, 5 : < \text{occupation} > \_ < \text{singer} > (x) \rightarrow < \text{performer\_musical\_artist} > (x, y).$$

5.3. Completeness Analysis of IOP Rule Learning

SHACLEARNER uses two tricks to significantly reduce the search space for IOP rules in PathFinding() of Algorithm 1, namely the prior Sampling() and the heuristic pruning used inside PathFinding() that uses the embedding-based scoring function. We conduct experiments to explore how these two pruning methods affect SHACLEARNER with regard to the quality and quantity of learnt rules. To do so we create three variants of SHACLEARNER as follows.

(-S+H): SHACLEARNER that does not sample and so uses the complete input KG in all components, including embedding learning, heuristic pruning and ultimate evaluation.

(+S-H): SHACLEARNER that samples but does not use heuristic pruning and so generates rules based on the sampled KG and evaluates rules on the complete KG.

(-S-H): SHACLEARNER that does not use sampling nor heuristic pruning. This system is an ideal IOP rule learner that generates and evaluates all possible rules up to the maximum length parameter. Since this method is a brute-force search, it cannot handle any real-world KG such as those we used in the performance evaluation (section 5.2).

(+S+H): SHACLEARNER with its full functionality.

Since we use only the small Poker KG for this experiment, we can handle the task without sampling or heuristic mechanisms. We use all 28 unary predicates as the target predicates. The first row of table 3 shows

Table 3
IOPSC performance of the ideal no-pruning SHACLEARNER variant on Poker, showing the percentage reduction in rules found by IOPSC intervals for variants with sampling and/or heuristic pruning

Learner/IOPSC	[0.1,0.3)	[0.3,0.5)	[0.5,0.7)	[0.7,0.9)	[0.9,1]
#(-S-H)	163	44	74	36	46
(-S+H)%	-10%	0%	-32%	-25%	0%
(+S-H)%	-93%	-93%	-12%	-11%	0%
(+S+H)%	-98%	-93%	-35%	-22%	0%

the number of IOP rules that are learnt by ideal, modified SHACLEARNER (-S-H) with no pruning, for all target predicates, separated into various IOPSC intervals. The latter rows show, for each variant, the percentage reduction in the number of rules found, relative to the first row. The last row corresponds to unmodified SHACLEARNER of Algorithm 1.

We observe that unmodified SHACLEARNER does not miss any rules of the highest quality, i.e., with $IOPSC \geqslant 0.9$. SHACLEARNER's pruning methods cause it to fail to discover more rules of lower quality, with the number of missing rules increasing as quality decreases. This is a reassuring property, since the goal of pruning is to improve the computational performance without missing high-quality rules. In real applications we will typically retain and action only the highest quality rules.

We observe that, unlike the other pruning variants, using heuristic pruning alone in (-S+H) does not uniformly increase in effectiveness with decreasing rule quality. This may be because using the complete KG for learning rules about all target predicates could harm the quality of the learnt embeddings.

5.4. Learning Trees from IOP Rules

Now we turn to presenting results for the trees that are built based on the IOP rules discovered in the experiments. We report the characteristics of discovered trees in Table 4. We use a value of 0.1 for the $TreeSCMIN$ parameter and show average TreeSC for each KG, along with the average number of branches in the trees and the average runtime building each tree. The number of trees for each KG is defined by the number of target predicates for which we have at least one IOP rule (see %SucTarget in Table 2).

Table 4
Tree-learning performance of SHACLEARNER on benchmark KGs

Benchmark	TreeSC	#branches	Time (hours)
YAGO2s	0.13	21	0.06
Wikidata+UP	0.19	21	0.1
DBpedia+UP	0.2	88	0.14

The results show the running time for aggregating IOP rules into trees is lower than the initial IOP mining time by a factor greater than 10. If, on the other hand, we wanted to discover such complex shapes from scratch it would be exhaustively time consuming due to the sensitivity of rule learners to the maximum length of rules. The number of potential rules in the search space grows exponentially with regards to the maximum number of predicates of the rules. The average number of branches in the mined trees are 50%, 31%, and 56% of the corresponding number of mined rules respectively from Table 2. Hence, by imposing the additional tree-shaped constraint over the basic IOP-shaped constraint, at least 44% of IOP rules are pruned.

For an example of tree shape learning, in the following, we show a fragment of a 39-branched tree mined from DBpedia by aggregating IOP rules in the experiments.

$$
\begin{aligned}
0.13 : <\mathsf{dbo:type}>\_<\mathsf{db:Song}>(x) \rightarrow \\
1 : <\mathsf{dbo:album}>(x, z_1)\wedge \\
<\mathsf{dbo:recordLabel}>(z_1, y_1)\wedge \\
2 : <\mathsf{dbo:album}>(x, z_2)\wedge \\
<\mathsf{dbo:producer}>(z_2, y_2)\wedge \\
1 : <\mathsf{dbo:album}>(x, z_3)\wedge \\
<\mathsf{dbo:genre}>(z_3, y_3)\wedge \\
3 : <\mathsf{dbo:artist}>(x, z_4)\wedge \\
<\mathsf{dbo:musicalArtist}>(z_4, y_4).
\end{aligned}
$$

Here, the first annotation value (0.13) presents the SC of the tree and the subsequent values at the beginning of each branch indicate the branch cardinality. This tree can be read as saying that a song has an album with a record label, an album with two producers, an album with a genre, and an artist who is a musical artist.

As can be seen here, there remains an opportunity for improving tree shapes for simplicity and easier interpretation by unifying some variables that occur in predicates that occur in multiple branches. We plan to investigate this potential post-processing step in future work.

6. Conclusion

In this paper we propose a method to learn SHACL shapes from KGs as a way to describe KG patterns, to validate KGs, and also to support new data entry. For entities that satisfy target predicates, our shapes describe conjunctive paths of constraints over properties, enhanced with minimum cardinality constraints.

We reduce the SHACL learning problem to learning a novel kind of rules, Inverse Open Path rules (IOP). We introduce rule quality measures IOP Standard Confidence, IOP Head Coverage and Cardinality which augment the rules. IOPSC effectively extends SHACL with shapes, representing the quantified uncertainty of a candidate shape to be selected for interestingness or for KG verification. We also propose a method to aggregate learnt IOP rules in order to discover more complex shapes, trees.

The shapes support efficient and interpretable human validation in a depth-first manner and are employed, for example, in an editor *Schímatos* [24] for manual knowledge graph completion. The shapes can also be used to complete information triggered by entities with only a type or class declaration by automatically generating dynamic data entry forms. In this manual mode, they can also be used more traditionally to complete missing facts for a target predicate, as well as other predicates related to the target, while enabling the acquisition of facts about entities that are entirely missing from the KG.

To learn such rules we adapt an embedding-based Open Path Rule Learner (OPRL) by introducing the following novel components: (1) we propose an IOP rule language which allow us to mine rules with open variables with one predicate forming the body and a chain of predicates as the head, while keeping the complexity of the learning phase manageable; (2) we introduce cardinality constraints and tree shapes for more expressive patterns; and (3) we propose an efficient method to evaluate IOP rules and trees by exactly computing the quality measures of each rule using fast matrix and vector operations.

Our experiments show that SHACLEARNER can mine IOP rules of various lengths, cardinalities, and qualities from three massive real-world benchmark KGs including Yago, Wikidata and DBpedia.

Learning shape constraints from a schema-free knowledge-base like modern KGs is a challenging task. The challenge starts from the formalism of constraints that determine the scope of knowledge that could be acquired. The next challenge, which is dependent on the first choice, is to design an efficient learning method. Dealing with uncertainty in the constraints and the learning process adds an extra dimension of challenge, but also adds to utility. A good learning algorithm should scale gracefully so that discovered constraints are relatively more certain than those that are missed. SHACLEARNER establishes a benchmark for this problem.

In future work we will validate the shapes we learn with SHACLEARNER via formal human-expert evaluation and further extend the expressivity of the shapes we can discover. We also propose to redesign the SHACLEARNER algorithm for a MapReduce implementation to handle extremely massive KGs with tens of billions of facts, such as the most recent version of Wikidata.

Acknowledgments

The authors acknowledge the support of the Australian Government Department of Finance and the Australian National University for this work.

References

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak and Z.G. Ives, DBpedia: A Nucleus for a Web of Open Data, in: *ISWC*, Lecture Notes in Computer Science, Vol. 4825, Springer, 2007, pp. 722–735.

[2] F.M. Suchanek, G. Kasneci and G. Weikum, Yago: a core of semantic knowledge, in: *WWW*, ACM, 2007, pp. 697–706.

[3] N. Noy, Y. Gao, A. Jain, A. Narayanan, A. Patterson and J. Taylor, Industry-Scale Knowledge Graphs: Lessons and Challenges, *Commun. ACM* **62**(8) (2019), 36–43–.

[4] E.S. Callahan and S.C. Herring, Cultural bias in Wikipedia content on famous persons, *Journal of the American Society for Information Science and Technology* **62**(10) (2011), 1899–1915.

[5] D. Vrandecic and M. Krötzsch, Wikidata: a free collaborative knowledgebase, *Commun. ACM* **57**(10) (2014), 78–85.

[6] H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), 2017. https://www.w3.org/TR/shacl/.

[7] Z. Huo, K. Taylor, X. Zhang, S. Wang and C. Pang, Generating multidimensional schemata from relational aggregation queries, *World Wide Web* **23** (2020).

[8] I. Boneva, J. Dusart, D. Fernández-Álvarez and J.E.L. Gayo, Semi Automatic Construction of ShEx and SHACL Schemas, *CoRR* **abs/1907.10603** (2019). http://arxiv.org/abs/1907.10603.

[9] P. Ghiasnezhad Omran, K. Taylor, S. Rodríguez Méndez and A. Haller, Towards SHACL Learning from Knowledge Graphs, in: *{ISWC2020} Posters & Demonstrations*, CEUR Proceedings, 2020.

[10] H. Knublauch, D. Allemang and S. Steyskal, SHACL Advanced Features, 2017. https://www.w3.org/TR/shacl-af/.

[11] L. Galárraga, C. Teflioudi, K. Hose and F.M. Suchanek, Fast rule mining in ontological knowledge bases with AMIE+, *The International Journal on Very Large Data Bases* (2015), 707–730. ISBN 1066-8888.

[12] L. Bellomarini, E. Sallinger and G. Gottlob, The Vadalog system: Datalogbased reasoning for knowledge graphs, in: *VLDB*, Vol. 11, 2018, pp. 975–987. ISSN 21508097.

[13] W. Fan, C. Hu, X. Liu and P. Lu, Discovering graph functional dependencies, in: *SIGMOD*, ACM, 2018, pp. 427–439. ISSN 07308078. ISBN 9781450317436.

[14] P. Ghiasnezhad Omran, K. Taylor, S. Rodríguez Méndez and A. Haller, Active Knowledge Graph Completion, Technical Report, ANU Research Publication, 2020.

[15] P. Ghiasnezhad Omran, K. Taylor, S. Rodriguez Mendez and A. Haller, Active Knowledge Graph Completion, in: *ISWC Satellite Tracks (Posters and Demonstrations, Industry, and Outrageous Ideas)*, 2020, pp. 89–94.

[16] P.G. Omran, K. Wang and Z. Wang, Scalable Rule Learning via Learning Representation, in: *IJCAI*, 2018, pp. 2149–2155. ISBN 9780999241127.

[17] Y. Chen, D.Z. Wang and S. Goldberg, ScaLeKB: scalable learning and inference over large knowledge bases, *The International Journal on Very Large Data Bases* (2016), 893–918.

[18] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in: *VLDB*, Vol. 1215, 1994, pp. 487–499.

[19] K. Taylor, Generalization by absorption of definite clauses, *The Journal of Logic Programming* **40**(2–3) (1999), 127–157.

[20] S. Staworko, I. Boneva, J.E. Labra Gayo, S. Hym, E.G. Prud'hommeaux and H. Solbrig, Complexity and Expressiveness of ShEx for RDF, in: *ICDT*, 2015, pp. 195–211. http://linkeddata.org/.

[21] M. Nickel, L. Rosasco and T. Poggio, Holographic Embeddings of Knowledge Graphs, in: *AAAI*, 2016, pp. 1955–1961.

[22] M. Nickel, V. Tresp and H.-P. Kriegel, A three-way model for collective learning on multi-relational data, in: *ICML*, 2011, pp. 809–816.

[23] P.G. Omran, K. Wang and Z. Wang, An Embedding-based Approach to Rule Learning in Knowledge Graphs, *TKDE* **33** (2021), 1348—1359. doi:10.1109/TKDE.2019.2941685. https://ieeexplore.ieee.org/document/8839576/.

[24] J. Wright, S. Rodríguez Méndez, A. Haller, K. Taylor and P. Omran, Schimatos: a SHACL-based Web-Form Generator for Knowledge Graph Editing, in: *ISWC*, 2020.

[25] N. Mihindukulasooriya, M. Rifat, A. Rashid, G. Rizzo, R. García-Castro, O. Corcho and M. Torchiano, RDF Shape Induction using Knowledge Base Profiling, in: *Annual {ACM} Symposium on Applied Computing, {SAC}*, Vol. 8, 2018, p. pages. ISBN 9781450351911.

[26] D. Fernández-Álvarez, H. García-González, J. Frey, S. Hellmann and J.E.L. Gayo, Inference of latent shape expressions associated to DBpedia ontology, in: *ISWC Posters*, Vol. 2180, 2018. ISSN 16130073.

[27] B. Spahiu, A. Maurino and M. Palmonari, Towards improving the quality of knowledge graphs with data-driven ontology patterns and SHACL, in: *Workshop on Ontology Design and Patterns*, Vol. 2195, 2018, pp. 52–66. ISSN 16130073.

[28] I. Boneva, J. Dusart, D.F. Álvarez and J.E. Labra Gayo, Shape designer for ShEx and SHACL constraints, in: *ISWC Posters*, Vol. 2456, 2019, pp. 269–272. ISSN 16130073.

[29] A. Cimmino, A. Fernández-Izquierdo and R. García-Castro, Astrea: Automatic Generation of SHACL Shapes from Ontologies, in: *ESWC*, Vol. 12123 LNCS, 2020, pp. 497–513. ISSN 16113349. ISBN 9783030494605.

[30] H.J. Pandit, D. O'Sullivan and D. Lewis, Using ontology design patterns to define SHACL shapes, in: *CEUR Workshop Proceedings*, Vol. 2195, 2018, pp. 67–71. ISSN 16130073.

[31] J.-E. Labra-Gayo, E. Prud'hommeaux, H. Solbrig and I. Boneva, Validating and describing linked data portals using shapes, *CoRR* (2017).

[32] I. Boneva, J.E. Labra Gayo and E.G. Prud'hommeaux, Semantics and validation of shapes schemas for RDF, in: *ISWC*, Vol. 10587 LNCS, 2017, pp. 104–120. ISSN 16113349. ISBN 9783319682877. http://www.w3.org/Submission/spin-modeling/.

[33] P. Pareti, G. Konstantinidis, T.J. Norman and S. Murat, SHACL Constraints with Inference Rules, in: *ISWC*, 2019.

[34] J. Corman, J.L. Reutter and O. Savković, Semantics and validation of recursive SHACL, in: *ISWC*, Vol. 11136 LNCS, 2018, pp. 318–336. ISSN 16113349. ISBN 9783030006709.

[35] R. Cattral, F. Oppacher and D. Deugo, Evolutionary Data Mining with Automatic Rule Generalization., *Recent Advances in Computers, Computing and Communications* (2002), 296–300.

[36] D. Dua and C. Graff, UCI Machine Learning Repository, 2017, archive.ics.uci.edu/ml Retrieved Nov 2019.

Author responses to the comments
Article title: Learning SHACL Shapes from Knowledge Graphs
Tracking Number: 2681-3895
Authors: Pouya Ghiasnezhad Omran, Kerry Taylor, Sergio Rodriguez Mendez, and Armin Haller

We appreciate the reviewers' comments which have assisted us to improve the manuscript. Please find below our detailed response to the comments made by the reviewers. Adjusted parts are highlighted. We published the executable code of SHACLEARNER in the shared folder of the paper.

Editor comments

	comments	responses
0.1	The reviewers suggest to better clarify the paper's contributions (especially with respect to the previous contribution, what's the delta?), which parts of the algorithm are previous work and which parts are contributions of the paper under review, whether pruning indeed removes only irrelevant entities and predicates and your definition of similarity and completeness.	We added the following clarifications: To section "1. Introduction": In summary, the main contributions of this paper are: – We introduce a new formalism called Inverse Open Path rules, that serves as a building block for more complex shapes such as trees, together with cardinality constraints and quality measurements; – We extend the Open Path rule learning method, OPRL [14, 15], to learn IOP (Inverse Open Path) rules annotated with cardinality constraints, while introducing unary predicates that can act as class or type constraints; and – We propose a method to aggregate IOP rules to produce tree shapes. To section 3.2: While the overall algorithm structure of SHACLEARNER is similar to OPRL [14], as is the embedding-based scoring function, the following elements are novel in SHACLEARNER: -OPRL cannot handle unary predicates while SHACLEARNER admits unary predicates both in the head and the body of IOP rules. -SHACLEARNER can discover and evaluate IOP rules with minimum cardinality constraints in the head of the IOP rule, while OPRL is effectively limited to learning the special case of minimum cardinality 1 for all rules. For this reason, the evaluation method of SHACLEARNER, Ev() differs from the OPRL evaluation module. -The aggregation module that produces trees out of learnt IOP rules, ready for translation to SHACL, is novel in SHACLEARNER.

		For : “whether pruning indeed removes only irrelevant entities and predicates and your definition of similarity and completeness “ please see new section 5.3.
0.2	The reviewers also suggest extending the analysis and discussion of the embedding component and consider a baseline for the evaluation, especially with respect to completeness and correctness.	We included such a comparison regarding different components of our system. Please refer to the new section “5.3 Completeness Analysis of IOP Rule Learning”. We also extended the presentation of the embedding component a little in new section 3.4 (and have improved the references to the paper where this is taken from throughout).

Reviewer 1 comments

	comments	responses
1.1	Then defines an open path (already defined in the tech report [13]) as a dependency of the form P(x,z_0) <- P_1(z_0, z_1), ..., P_n(z_{n-1}, y) which I do not fully understand because the quantifications are not explicitly given. The authors say that this rule contains free variables but I do not know which variables are free. x? z_0 ? Y?	We revised it as follows in section 2.2. $P1(z0,z1) \wedge P2(z1,z2) \wedge \ldots \wedge Pn(zn-1,y) \rightarrow \exists x: Pt(x,z0)$ Here, Pi is a predicate in the KG, each of {x,zi,y} are entity variables, and all free variables are universally quantified at the outside. We call a variable closed if it occurs in at least two distinct predicate terms, such as z_0 here, and otherwise it is open. If all variables of a rule are closed then the rule is closed. An OP rule has two open variables (y and x). [14] uses OP rules since they imply the existence of a fact, like spouse(x,y)→∃z child(x,z)[11]. Unlike CP rules, OP rules do not necessarily form a loop, but a straightforward variable unification transforms an OP rule to a CP rule, and every entity-instantiation of a CP rule is also an entity-instantiation of the related OP rule (but not vice-versa).
1.2	Here is a suggestion of how one could possibly discover rules discarded by the above mentioned approximations: take a small KG and apply SHACLEARNER on it w/o any approximations, and with the approximations. Do you get the same rules ? If yes, then nothing is	Thank you for this very helpful suggestion. We have newly included such comparisons regarding different components of our system. Please refer to the new section “5.3. Completeness Analysis of IOP Rule Learning”.

	proved (SHACLEARNER could still miss important rules on another KG). If no, then you have an indication on the rules that the approximation does not allow to discover. Is it possible to quantify the precision / completeness of the algorithm? Can you compute the probability that a rule satisfied in the KG is indeed discovered ?	
1.3	In particular, the zero-or-one cardinality is very important while modeling, but SHACLEARNER never generates rules with zero-or-one cardinality.	A rule or shape with cardinality zero would indicate that the shape does not have any reason to be, as there is no witness for it in the KG. Another way to see this is that a different rule (shorter, but with a witness in the KG) would be discovered instead with cardinality 1, to take the place of a cardinality 0 rule. In another way to look at this, consider the IOP rule with cardinality 1 and IOPSC 0.6. This means there are 60% of instances of the rule where cardinality 1 holds, and 40% of instances where it does not (ie cardinality zero).
1.4	How relevant are the discovered shapes? Can you describe use cases or use scenarios in which precisely these shapes are relevant?	Please refer to the new section "3.7.1. Tree Shapes are Useful for Human Interaction".
1.5	In the Conclusion the authors do motivate the use of SHACL shapes for knowledge graphs, but are the shapes discovered by SHACLEARNER relevant for any of these tasks? For instance, are rules discovered by SHACLEARNER relevant for generating forms for data completion? In order to complete data one needs a reliable and approved schema. Can automatically discovered rules be used to decide that some data is missing in the graph?	Yes, the discovered SHACLs are fed to Schimatos to generate forms based on them. Based on the confidence degree of a learned shape, correspondent data input entries with a degree of necessity have been offered in the forms. Based on the shapes and given data, Schimatos can suggest the missing facts to users to fill the gaps. While we agree that "a reliable and approved schema" would be best, in practice they do not exist for many large KGs and it is impractical to design them post-hoc. We could imagine that human curation could be used to check the shapes we discover before import into Schimatos, but so far we have not found this necessary and choosing very high quality rules is sufficient.

1.6	Also, SHACLEARNER is based on discovering paths of some length that are then combined. Does it often occur in practical applications to use paths in constraints? IMHO the most frequent case is for the shape to describe the immediate neighborhood of the node, or maybe up to distance 2 or 3. What use scenarios justify the use of such a complex algorithm as SHACLEARNER that explores the huge search space of all paths up to some length? When is the exploration of just the immediate neighborhood not enough for discovering a 'good' shape?	Agree. In our experiment, we use IOP rules with up to three body atoms. Although our given algorithms, including the sampling, can be extended to any lengths, the IOP rules with up to 3 body atoms would be enough in practice. In general, other rule learning algorithms like AMIE+ and ScaleKB, use two body atoms in the body for large KGs while they do not have any existentially quantified variables that make the search more complicated.
1.7	p. 9 l. 27-34, comparison with [31]: I wouldn't say that [31] is "partial attempt" to provide logical foundations of the semantics of SHACL. It is I think quite complete, and in any case a more complete proposal of the semantics than the rules presented in the paper under review. Btw it could be interesting to compare the rules you propose with the formalization proposed in [31].	Agree and we revised it as follows. There are some attempts to provide logical foundations for the semantics of the SHACL language, including [34] that presents the semantics of recursive SHACL shapes. By contrast, in our work, we approach SHACL semantics in the reverse direction. We start with logical formalisms with both well-defined semantics and motivating use cases to derive shapes that can be trivially expressed in a fragment of SHACL.
1.8	- p. 9 , first paragraph of Section 4. The related works [8,22-25] are qualified "without logical foundation" and their output formalisms are implicitly qualified as not "well-founded". While I myself like to use logic formalisms in order to define things w/o ambiguity, I wouldn't say that other kinds of definitions are not well-founded. As the main author of [25] I claim that the output of the shape inference algorithm there is very precisely qualified (see also the companion work [**]): it is a precisely defined generalization to SHACL rules of the neighborhoods of a set of randomly chosen sample nodes. No probabilities, no heuristics, but an outspoken random sampling with clearly identified use	Some of the mentioned methods handle real-world KGs to some extent, like [22] (now [24] in this update), while their formalism is different from ours. In this case, [22] cannot discover the shape with a sequence of predicates (nested conditions), so comparing these methods is not feasible. Some other techniques do not contain a scalable method that can be used for comparison purposes. Previous [25] is now numbered 8 in this update. In [8] the authors provide an interactive framework to define SHACL shapes with different complexities, including nested patterns that are similar to the trees that we use. A formalisation of the approach is given, but there are no shape quality measures that are essential for large scale shape mining. Because the paper does not provide a system that discovers patterns from massive

	scenarios: help expert users to write schehmas, help novice users to get some idea of the shape of the data.	KGs, we can not deploy their method for comparison purposes.
1.9	- p. 2, right, l. 46: it is said that the rule forms a "loop". The standard definition of a loop I know of is a path that starts and ends with the same element. It is not the case here. What kind of loop is it ?	Yes it is the same, although we allow the connecting variables to take either the subject or object argument in a predicate. So that sometimes a predicate needs to be reversed to be read directly as a loop, but this is a syntactical consideration. Consider the closed path rule like, P_1(x,y) :- P_2(x,z), P_3(z,y). There is a loop as follows: 1. start from variable x 2. Traverse from x to z by P_2 3. Traverse from z to y by P_3 4. Traverse from y to x by P_1 5. Ended up in the same element y
1.10	- Definitions of OP and IOP rules, I found it a bit disturbing to distinguish the variable y from the z variables, as in the definitions of satisfies and support, SC and HC y is also existentially quantified. Later on I could see that y is relevant for the cardinality. Maybe this could be mentioned right away, to clarify why it treated differently from the z's.	Please refer to point 1.1.
1.11	p. 12, right, l. 25: what you call %SucRate was previously called %SucTarget	Revised
1.12	I'm curious about the matrix representation used to explain the efficiency of the algorithm. With the size of the KBs considered, you have matrices of size n x n with n=3 000 000. Can it even fit in RAM (even if you represent 1 Boolean value with 1 bit) ? Do you use sparse matrices ? What kind of matrix multiplication algorithm is being used ?	Yes. We use a sparse matrix representation in python, namely dok_matrix, csr_matrix. We use different formats of sparse matrix representations in the algorithm based on the efficiency of the specific matrix manipulation that we need. For multiplication, we use the provided algorithm in “scipy.sparse".
1.13	- Clearly state what are the new contributions of this paper and what comes from previous work,	Over the paper we stated that the components of SHACLEARNER are sampling, heuristic path finding, evaluation, and tree search. The sampling

	especially in Section 3 and the SHACLEARNER algorithm.	is commonly used in rule learning methods [13, 15] as we describe it in algorithm 2. The embedding is obtained from RESCAL [20, 21] as it can provide an extensive representation of predicates and entities as it shows in previous heuristic rule learners like [22]. The heuristic path finding (scoring function) is similar to OP rule learner proposed in [13] due to the similarity between the paths that are being used for OP and IOP rules. In the evaluation and tree search parts, SHACLEARNER is quite different from OPRL, as we summarized in point 0.1. Please refer to point 0.1.
1.14	- State precisely what are the external components that are used by SHACLEARNER(eg RLvLR, RESCAL) and what are their properties.	The main components of SHACLEARNER are sampling, heuristic path finding, and evaluation. The sampling is commonly used in rule learning methods[14, 16] as we describe it in algorithm2. The embedding is obtained from RESCAL [21, 22] as it can provide an extensive representation of predicates and entities as it shows in previous heuristic rule learners like [16]. The heuristic path finding (scoring function) is similar to OP rule learner proposed in [14] due to the similarity between the paths that are being used for OP and IOP rules. In the evaluation part, SHACLEARNER is very different from OPRL. Throughout the paper we have clarified novel and reused components in this revision. Section 3.2 has a new paragraph that explicitly states the novel elements of SHACLEARNER (see “While the overall algorithm......is novel in SHACLEARNER”). Please refer to 0.1.
1.15	- equation (5): what are the formal properties of the scoring function ? What precision does this scoring function have, i.e. what guarantee does the value computed in (5) give on the actual probability of P_0(e_1, e_2) ?	The scoring function is a heuristic function that we do not have formal properties or guarantees, while in the extended experiment with a complete and correct Poker KG, we show it works well, especially for rules with high IOPSCs. Such scoring functions are commonly used in embedding-based Knowledge Completion methods.
1.16	- what is similarity (mentioned on p. 6 l.48) ?	The vector that represents argument P^1_2 should be close to the vector that represents P^2_1. We

	“The predicate arguments that have same variable in the rule should have similar argument emdeddings. For example we should have the following similarities, P^2_t ~ P^1_1 and P^1_2 ~ P^2_1.”	use the Frobenius norm to measure such distance between two vectors. The assumption is that if two arguments share the same variable in a rule, it is likely that those arguments share many entities that cause the vector representations of them similar. This discussion is only a summary in this paper and the reader is referred to the source[14] for detail.
1.17	- Explain what is the sampling or at least what are its formal properties so that one can judge whether sampling would remove relevant parts of the KG.	We added a new pseudocode and explanation for sampling as section “3.3. sampling”.
1.18	- Qualify and/or quantify the output of the SHACLEARNERalgorithm. What kinds of paths might not be discovered ? What is the probability that an actual frequently occurring path is discovered by the algorithm ?	Based on the extended analysis, the algorithm is likely to miss the paths with low IOPSCs. We provide such discussions in the new section “5.3. Completeness Analysis of IOP Rule Learning”.
1.19	- Identify use scenarios: what use cases can be supported by SAHCLEARNER, in particular in comparison with simpler algorithms ?	This algorithm can discover shapes with different properties and certainties from massive KGs. After discovering such patterns, we feed them to Schimatos to generate interactive forms to assist the user in filling the gaps.
1.20	Additional citation [**] Semi Automatic Construction of ShEx and SHACL Schemas Iovka Boneva, Jérémie Dusart, Daniel Fernández Álvarez, Jose Emilio Labra Gayo.	Done. We added the following part to the related work. In [8] the authors provide an interactive framework to define SHACL shapes with different complexities, including nested patterns that are similar to the trees that we use. A formalisation of the approach is given, but there are no shape quality measures that are essential for large scale shape mining. Because the paper does not provide a system that discovers patterns from massive KGs, we can not deploy their method for comparison purposes.

Reviewer 2 comments

	comments	responses
2.1	(i) Is RESCAL the best model for learning embeddings to detect paths?	RESCAL captures complex interactions between entities and predicates. More recent embedding-

		based link predictors such as HOLE and TuckER embeddings that are more complex and compact than RESCAL. The later embeddings' compactness is beneficial for link prediction, but it is counterproductive for our goals, since we are employing embeddings in a heuristic function. Furthermore, RESCAL has been shown to be effective when used as a heuristic for mining logical axioms, which is exactly what we need [14, 16, 23].
2.2	(ii) Could you show it empirically, comparing the results with other embedding methods?	Such an empirical study that investigates different kinds of embedding representation for the propose of learning rules is an interesting idea. Still, it is challenging due to the demand to change other parts of the system, including the scoring function or even sampling. Hence, we did not peruse such investigations in this paper.
2.3	(iii) Which is the setting of hyperparameters that allow you to achieve the best performance?	We uploaded the system parameters to shared folder of the paper. In the following we stated those parameters as well. For the sampling process, SHACLEARNER picked at most 15 neighbors of an entity and set the maximum size of each sample to 800 entities. For the embedding generation, we used RESCAL with the sizes of entity embeddings set to 100. We found through experiments (parameter tuning) that such a configuration provides SHACLEARNER relatively optimal performance.The number of possible candidate rules grows significantly as the lengths of rules increase, so does the runtime for mining. For our system, we use a maximum length of 4.
2.4	Minor changes	Revised.

Reviewer 3 comments

	comments	responses
3.1	- Originality: Framing the problem of learning SHACL Shapes as a rule learning problem with paths using Inverse Open Paths can be considered as original. However, it will be important explicitly identify which are the novel contributions of this paper and what is being reused from the existing works.	The main components of SHACLEARNER are sampling, heuristic path finding, and evaluation. The sampling is commonly used in rule learning methods[14, 16] as we describe it in algorithm2. The embedding is obtained from RESCAL [21, 22] as it can provide an extensive representation of predicates and entities as it shows in previous heuristic rule learners like [16]. The heuristic path finding (scoring function) is similar to OP rule learner proposed in [14] due to the

		similarity between the paths that are being used for OP and IOP rules. In the evaluation part, SHACLEARNER is very different from OPRL. Throughout the paper we have clarified novel and reused components in this revision. Section 3.2 has a new paragraph that explicitly states the novel elements of SHACLEARNER (see “While the overall algorithm......is novel in SHACLEARNER”). Please refer to 0.1.
3.2	- Significance of results: One of the main weaknesses of this paper is the evaluation. The hypothesis being validated is vague with claims such as “handle the task satisfactorily”. Evaluation needs to be improved to include (a) a reasonable baseline to compare with (b) some evaluation of the correctness of the shapes learned (maybe manually on a sample) and/or the usefulness of them in a real life use case (c) some qualitative analysis. Detailed comments are provided below.	We include such comparison regarding different components of our system. Please refer to the new section “5.3. Completeness Analysis of IOP Rule Learning”.
3.3	- This section puts some emphasis on the incompleteness of the knowledge graphs. Though the path rules can be used to infer new triples, the focus of this paper is on learning SHACL Shapes. It might be good to somehow relate this paragraph about incompleteness to the overall story.	We added the following part to the introduction. Both kinds of errors can be highlighted for correction by the careful application of schema constraints. However, such constraints are commonly unavailable and, if available, frequently violated in a KG for valid reasons, arising from the intended data-first approach of KG applications.
3.4	- As SHACL Shape learning is the main topic of the paper, it would be beneficial if the paper provided a brief description of all different constraint types such as value type, cardinality, value range, string-based, property-pair etc and state explicitly which ones are covered by the proposed approach (nodeKind and minCount?). - It will be also interesting to identify which constraint types can not be	We added to end of section 2.3. SHACL shapes. Various formalisms with corresponding shapes have been proposed to express diverse kinds of patterns exhibited in KGs, such as k-cliques, Closed rules(CR) [11] (that include closed path rules), Functional Graph Dependency (FGD) [13], and trees [12]. CRs are used for inferring new facts. FGDs define constraints like the type of entities in the domain and range of predicates, or the number of entities to which an entity can be related by a specific predicate. These constraints are deployed to make the KG consistent.

	fundamentally addressed using path-based approaches, for example, value ranges or property-pairs? It might also help to validate the claim “common underlying form of all these shapes is the path”.	Regardless of differences amongst these formalisms, they share a key feature in their syntax. The building block for expressing all of these shape constraints is a sequence of predicates.
3.5	- It would help the reader if you can include an explicit list contributions at the end of the introduction using the common practise of “the main contributions of this paper are: (a) … (b) … (c) …”. This will help the reader to understand what are the novel ideas and what is being reused.	Please refer to point 0.1.
3.6	- Even after reading the paper completely, I didn’t quite understand the need for these additional unary predicates. What is the motivation for this? The IOP definition handles the binary predicates (as you do in Yago2). Is there any difference if you used dbo:type(x, dbo:Singer) as the body of your IOP instead of the unary predicate? - On a separate note (if unary predicates were really necessary), why rdf:type and P31/P279 properties were not used for this? A discussion on motivation for this decision and the selection criteria will be useful to the reader. If rdf:type was used, it would have prevented unconventional target classes in SHACL shapes such as \`sh:targetclass class:_;\`. - Isn’t the notation P(e,e) for unary predicates conflict with reflexive properties in the KG. For me personally, the idea of representing unary predicates as self-loops is counterintuitive. Also in the first paragraph of Section 2, P is used to denote both predicates and classes (where P is a class or a data type). To avoid confusion, it	The difference between dbo:type(x, dbo:Singer) and dbo:type_ dbo:Singer (x) is in the former one dbo:Singer presents as an entity while the latter one dbo:Singer presents as a predicate. To explain the rationale behind this kind of modeling we added the following part to section 5.1. Since real-world KG models treat predicates and entities in a variety of ways, we require a common representation for this work that clearly distinguishes entities and predicates. We employ an abstract model that is used in Description Logic ontologies, where classes and types are named unary predicates, and roles (also called properties) are named binary predicates. The modelling of the unary predicate as self-loop is inspired by the traversing graph. In each grounded path, we start from a node (entity) and move by choosing the first predicate of the path. This predicate could be a binary predicate that drives the agent to a new entity or could be a unary predicate (self-loop) that causes the agent to remain in the first entity. With regards to this scenario, modelling a unary predicate as a self-loop seems natural. We use the self-loop as syntactic sugar. We do not remove any facts but just add some new facts that able us to mine IOP rules. All mined rules can be presented in the original way. Hence, this modelling could be considered as an internal process of rule learning. For example, consider the following rule, <dbo:type>_<db:Album>(x,x)→<dbo:singer>(x,z).that

	might be better to use another symbol such as C.	can be presented in the original modeling as follows,<dbo:type>(x,<db:Album>)→<dbo:singer>(x,z).
3.7	- Please review your example in Figure 2 with the semantics of SHACL specification at https://www.w3.org/TR/shacl/#core-components.	Revised.
3.8	- Is Lemma 1 used in any part of the process? Why is it relevant?	This Lemma is relevant since it shows if we discard a rule with a length regarding its low IOPSC (v1), we do not need to check the extension of this rule with more head atoms since those rules' IOPSCs are equal or less than v1. Hence the process of pruning such rules does not harm the completeness of the rule learning.
3.9	- In 3.2, please provide details on the exact strategy/algorithm used for pruning the KG. Is it done once per each gold predicate? What is the avg size of the fragment of KG used for rule mining? This information is relevant as one of the claims of the paper is to handle massive KGs compared to the other approaches.	We added a pseudocode and explanation for sampling as a new section, 3.3. Besides, at the end for the ultimate evaluation of IOP rules we do not use the sampled KG, but we use the original KG. Hence, the relevance or quality of discovered IOP rules are not dependent on the sampled KG.
3.10	- Looking at the embedding based approach, one potential baseline would be to do the PathFinding but using only the symbolic information found in the graph. This might help to improve the evaluation.	Agree. We added such baseline. Please refer to the new section "5.3. Completeness Analysis of IOP Rule Learning".
3.11	- 3.2, there is a typo in predicate argument similarities. I assume the second one should be P12 and P21 not both P12.	Revised.
3.12	- Is there a novelty in the path finding approach or is it reusing what's defined in [13]?	It is the same as AKGC.
3.13	I believe the aspect of knowledge graph completion and the aspect question answering etc. are very interesting and could be important contributions of this paper. One of the requirements for that would be	Agree. We already use it through Schimatos. But conducting such an experiment with human users in the loop is quite challenging, so we decided not to investigate those parts in this paper and remain the scope to statistical evaluation based on the already existing rule learning systems in the literature.

	making them part of the evaluation. Can the evaluation be extended to cover some knowledge base completion or HCI aspects?	
3.14	- I think related work is too brief. For example for [22-25], it would be interesting to at least discuss what is the approach they use, what are the datasets they used, what are the differences with the proposed approach.	Some of the mentioned methods handle real-world KGs to some extent, like [22], while their formalism is different from ours. In this case, the [22] cannot discover the shape with a sequence of predicates (nested conditions), so comparing these methods is not feasible. Some other techniques do not contain a scalable method that can be used for comparison purposes.
3.15	- If there is no fair comparison out there, I guess you will at least have to provide a good baseline in order to validate your hypothesis.	We extended the experiments to compare our system to ideal complete rule learners and other different baselines.
3.16	- “handle the task satisfactorily”, “satisfactory performance” sound opinionated and not objectively verifiable. It would be good to formulate this with respect to the metrics identified.	We believe we show the satisfactory performance of our system via the quality, quantity, properties, and running time of our system on different real-world KGs. We also added the baseline comparison in section 5.3. to support this claim.
3.17	- “can be applied in practice” - how are you demonstrating this? Have you used this in any feasibility study in a real-world use case? I guess the trees learned (Sec 5.2) can be used for a qualitative analysis with a real use case.	By providing the output shapes as supplementary data and real examples in the experiment section, we show that the learnt shapes applicable. For our future work, we plan to conduct some experiments via using the learnt shape via Schimato tool and human users for qualitative analysis.
3.18	- As mentioned previously, I don’t understand the motivation for converting binary type assertions to unary relations in Section 5.0.1.	Please refer to point 3.6.
3.19	- in +UP, what is the intuition or reason for selecting dbo:type / dbo:class (DBpedia) and occupation_P106 (Wikidata) as unary predicates? What was the selection criteria among all other predicates with classes as their argument (for example, Wikidata has many of them)?	As we discussed in point 3.6, we use this syntactic sugar to present classes as unary predicates. We choose these predicates since the entities present class, and those classes are relatively intuitive. Hence, the produced SHACL shape could be readable for humans. We consider extending this experiment by considering a user study, so human readability of the shapes are important.

3.20	- Similarly, what’s the reason for using only binary predicates with YAGO2? What are the different characteristics of the KGs that motivated you to make these decisions?	To show the system's capability to handle binary predicates as well as unary one’s, we use binary predicates of YAGO2. Besides, in YAGO2, there are less obvious class concepts to form the unary predicates.
3.21	- For completeness and readability, it would be good to include a one liner about rationale for the specific minimum threshold (0.1 and 0.01) values for IOPSC and IOPHC though it might be discussed in detail in [10].	We use these numbers as these numbers are commonly used in the KG rule learning literature, including AMIE+ and RLvLR. The numbers are quite low, so most week rules are also covered by choosing such a low threshold. By discovering more rules, we do not lose anything, in a later stage for using the shapes, we could consider any higher threshold to consider the adequately confident rules.
3.22	- Analysis of Figure 4 and Figure 5 should be written with more details as it is one of the most interesting parts in the evaluation. I believe it is useful to first establish what high/low IOPSC and IOPHC indicates as discussed in 5.1.1. and then explain the charts relating to those. For example, “In the left-hand chart we observe a consistent decrease in the proportion of quality rules as the IOPSC increases.” has to be further discussed.	We added the following parts to section 5.2. For Fig 4: Over all benchmarks, the majority of learned rules have lower IOPSC and IOPHC. This is expected because the statistical likelihood of poor quality rules is much higher. Indeed, we show experimentally in section 5.3 that our pruning techniques, that are necessary for scalability, prune away predominantly lower quality rules. With respect to IOPSC, proportionally more quality rules are learnt from DBpedia than from the other KGs, with Wikidata second, ahead of YAGO2s. This phenomenon might be caused by the way that we select the target predicates: for Wikidata and DBpediawe handpick unary predicates that represent classes, while for YAGO2 we consider all binary predicates. For Fig 5: The distribution of mined rules with respect to their cardinalities is shown in Figure 5. We observe that the largest proportion of rules has a cardinality of 1, as expected, as they have the least stringent requirements to be met in the KG. We observe an expected decrease with greater cardinalities as they demand tighter restrictions to be satisfied. YAGO2 demonstrates a tendency towards higher cardinalities than the other KGs, possibly a result of its more curated development.
3.23	- Also in Figure 5, what can we learn from this chart? Is this	The observation of this experiment supports the property of Cardinality that we mentioned in section

	because of the characteristics of the domain such that there are more properties with less cardinality? Or is this because of the characteristics of the mining that it is easier to find rules that are defined as the lower bound? Furthermore, for practical use, you will have to pick one cardinality rule, isn't it?	3.1. "While we have a rule with a cardinality,c1, we also have rules with lowercardinalities1,...,(c1−1)but their IOPSCs should be as good or better than the rule withc1cardinality. This statement is quite intuitive since cardinality expresses the lower bound number of occurrences of the head." In practice, all cardinalities regarding a rule could be helpful. For example, consider the information about the children of a person. One child constraint has the highest IOPSC while the two children constraint has lower and so on. Now consider we want to get input regarding the children of a person, we could alter the necessity of each child with different degrees.
3.24	- The discussion in 5.1.1. touches the issue of concept hierarchy such that some rules apply to all human (and thus applies to all it's subtypes such as singers) and some rules are very specific to singers, for example. It would be interesting to discuss how this aspect of hierarchy or inheritance can be incorporated into your proposed solution.	Agree. By processing the learnt IOP rules we could construct a set of hierarchy axioms like T-box subsumption axioms. However, we plan to investigate such studies in our future works.
3.25	- A qualitative evaluation/discussion of the usefulness of discovered trees will be beneficial for the paper.	We consider to do so as our future works and plan to conduct some evaluation and experiments with human users.
3.26	- "IOPSC effectively extends SHACL with shapes, representing the quantified uncertainty of a candidate shape to be selected" - do you plan to represent this information within the SHACL shape itself as metadata? It would be interesting to see an example of such representation.	Yes we plan to establish a rule hub that includes the SHACL shapes for different public KGs.
3.27	- Is this functionality already incorporated in schimatos.github.io?	Yes, we deployed the learned shapes via Schimatos and used if in our funded project.
3.28	- It would be good to include some lessons learned / challenges/limitations of the approach in the conclusion.	We added the following part to the conclusion section. Learning shape constraints from a schema-free knowledge-base like modern KGs is a challenging

		task. The challenge starts from the formalism of constraints that determine the scope of knowledge that could be acquired. The next challenge, which is dependent on the first choice, is to design an efficient learning method. Dealing with uncertainty in the constraints and the learning process adds an extra dimension of challenge, but also adds to utility. A good learning algorithm should scale gracefully so that discovered constraints are relatively more certain than those that are missed. SHACL EARNER establishes a benchmark for this problem.